Software Design For a
Fault-Tolerant Communications Satellite

G. Ken Hunter (1) and Neil C. Rowe (2)

(1) Information Management Office
Naval Security Group Activity, Fort Meade
9800 Savage Road
Fort Meade, MD 20755 USA
(2) Code CS/Rp, U.S. Naval Postgraduate School
Monterey, CA 93943 USA

(410)688-2515, (831) 656-2462
gkhunter@newsguy.com, rowe@cs.nps.navy.mil

Abstract

We describe a design of fault-tolerant features for the PANSAT communications satellite, a design which can address a wide variety of possible faults. We discuss system errors, program errors, and data errors, each subdivided into a variety of types. We discuss "acceptance tests" that can be used to detect faults, and the appropriate remediation methods for each type.

1. Introduction

Since 1989, the Space Systems Academic Group at the Naval Postgraduate School (NPS) has been developing the Petite Amateur Navy Satellite (PANSAT). PANSAT is an experimental, store-and-forward communications satellite developed and constructed almost entirely at NPS. Classified as a microsatellite, the spacecraft weighs 150 pounds and is nineteen inches in diameter. Using spread spectrum modulation, the satellite will provide email, binary file transfer, and point-to-point communications to amateur radio (HAM) users worldwide. The satellite was to be launched from the Space Shuttle and become operational in October 1998.

The PANSAT project is funded by the Navy Space Systems Division (N63). Primarily, the goal of this project is to be an educational tool for military officers as well as a learning experience for the Space Systems Academic Group. However, PANSAT also serves a significant secondary purpose. Currently, the military uses an eclectic, yet expensive, group of satellites for its communications. Many of these satellites are leased, and most of those that are not are quickly approaching their programmed lifetime. Replacing the satellites under the current architecture is expensive. Therefore, the Navy is evaluating the communications capabilities of inexpensive, yet dependable, small satellites with the idea of using them to augment or replace the current system. With this in mind, the PANSAT project is implemented with a minimum of cost, meaning that hardware resources are limited and redundant features are minimal. The inexpensive PANSAT satellite will be evaluated in an established communications environment to see if it can provide the reliable functionality that the military requires. If the experiment proves successful, PANSAT may become the cornerstone for a new military satellite communications network.

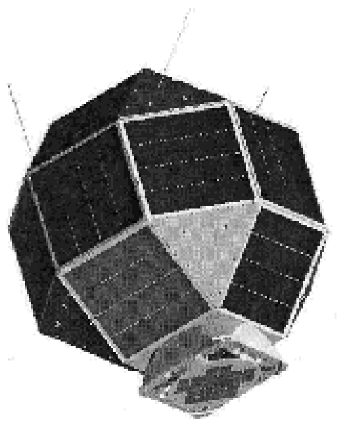

Figure 1: PANSAT

One of the difficulties with operating satellites, and one that severely impacts reliability, is that all problems which might arise with the unit must be detected and handled either remotely or by the spacecraft itself. Despite this difficulty, there are means of obtaining a reliable system. PANSAT′s counteractive goals of reliability and low cost do make detecting and handling errors even more of a problem to implement. Typically, to ensure a reliable space system, subsystems are implemented redundantly. However, this dramatically raises the costs, a luxury that cannot be afforded in the PANSAT project.

Space itself compounds errors. In space radiation is much stronger and has more impact on the operations of digital mechanisms. This effect, commonly referred to as "space anomalies", flips bits in digital circuits. The effects could range from none to crashing the entire system. Another feature of operating in space is that if one element on the satellite fails, it cannot be replaced. However, the entire mission should not be terminated because of a single part failure. Rather the system should be able to respond and adapt to overcome the failure using the resources available.

Fault tolerance is the means for a system to handle such errors or problems that might occur during operation. Developing a complete and proper fault tolerance plan should enable the system to gracefully detect and handle every type of problem that might arise during operations [Aviezienis, 1997; Somani and Vaidya, 1997]. Many different methods are involved in fault tolerance, but only the areas pertaining to the particular situation of the PANSAT implementation need to be identified here [Lee and Anderson, 1990]. While some fault tolerance techniques may have been applied to previous microsatellites, none has instituted as comprehensive a plan as designed for PANSAT. Additionally, this is the initial implementation of the Watchdog concept, developed for PANSAT in order to conduct system evaluations.

2. System Description

Before a fault tolerance plan can be developed, a survey of the resources available to the system needs to be conducted. PANSAT uses a space-rated Intel 80186 central processing unit (M80C186XL). While this is an older processor, it is space-hardened, meaning that it is resistant to space anomalies. Furthermore, the 80186 is inexpensive and fully capable of handling the workload expected of it. The spacecraft has two processors; the second processor merely monitors the first. If the first suffers a complete failure, when it shuts down due to some other independent power control system turning off the processor, the second processor takes over for the first.

The system has 512 kilobytes of working memory. This is error-checking volatile random access memory (RAM). There is no redundancy of working memory; however, this memory bank uses error-correcting Hamming codes to correct single bit flips. These codes can detect two bit flips, but not correct them. For three of more bit flips, the errors may not be detected. For long term storage, there are two four-megabyte memory banks. These memory banks are also volatile RAM, however they do not use an error correction scheme. Initially, these two banks will be set up mirroring each other. If data is lost in one bank, the other bank may still contain it. However, if the two banks disagree about the contents of the data, the system will be unable to determine which data bank contains the correct version. The data stored in these memory banks is communication data such as email and binary files.

In addition the four-megabyte storage, there are two 512 kilobyte non-volatile flash memory banks. Unlike the other memory banks, if the power is temporarily lost on the satellite, these banks will not lose their contents. Both working memory and the large storage banks will be completely erased even for a minuscule power outage. As the large store banks, these banks will be initially mirroring each other. These memory banks will be used to store telemetry and other important data. If experiments prove that the mirror memory configuration is not required, mirroring is turned off by the ground station. Without mirroring, the memory banks will work in tandem, doubling the storage memory capacity of the system. The working memory, however, will be unaffected.

The operating software for PANSAT will not be launched with the satellite. Rather, only a small bootstrap program is located in the system′s read-only memory (ROM). This bootstrap is set up to establish contact with the ground station, then have the operating system and functional programs uploaded to the satellite. After uploading, the programs are executed. Only then will PANSAT become an operational communications facility.

This uploading software concept has the benefit of being able to upload new versions of the software as required, allowing bug fixes or incorporation of additional features. Unfortunately, if the satellite resets due to a power loss or some unforeseen error, all the working programs must be reloaded from the ground station. In fact, if one program is unexpectedly terminated, then that one program needs to be re-uploaded. Termination means purged from working memory and no other copy of a program exists onboard the satellite.

The operating system is a multi-threaded kernel made by the BekTek Corporation called SCOS. All the satellite functional programs execute as threads on top of the operating system. The threads can communicate with each other using data streams, which are sent via SCOS. Of course, since only one processor is actually operating, the multi-threaded environment is obtained by interleaving the thread executions.

The main and largest thread running on PANSAT will be the User Services module. This is the program that actually performs the communications interface functions for the satellite. Most of the other threads support the operation of, or interface with, User Services. The scope of the fault-tolerance plan defined later is centered around, and in terms of, this main software module.

Some redundancy is obtained in the solar panels, sensors and batteries. If one solar panel fails, it will be discounted by the system. If more than a couple of solar panels fail, however, the satellite will not be able to generate enough power for the requirements. Thereafter, PANSAT′s performance would be sporadic at best. If one of the two battery packs fails, the life of the spacecraft is not just halved, but reduced exponentially. This is due to the life cycle of the batteries depending on full charges and discharges. One battery pack would not be able to be fully charged before it is started to be drained, thus is would not get the full cycles necessary for a healthy battery. Finally, failed sensors will simply have to be ignored and not used in telemetry analysis [Bible and Sakoda, 1998].

These hardware resources listed do provide some fault tolerance, in that most of the subsystems have backups that can take over if the first element fails. However, if an error less than complete failure occurs, the system can not gracefully recover from it. Because that level of error handling is not furnished in hardware, it is necessary to provide that functionality in software, which is the scope of the plan defined herein.

3. Previous Work

Some common methods for fault tolerance cannot be used for PANSAT due to hardware limitations. The method used most often is called N-version. In this method, several different versions of the same software are made in complete independence and isolation from each other. The results from each are then compared. The answer that has the most matches is assumed to be the correct one. Comparing answers this way is called voting. Voting not only can be used to compare N-versions, but also the exact same software run on different hardware. For instance, running the same program on multiple processors simultaneously could provide different results if one processor was faulty [Kreutzfeld, 1997]. PANSAT cannot use N-version or voting because one processor is running at a time. Also N-version is impractical because of the limited amount of working RAM, insufficient to spend on multiple versions of the same program.

Another commonly used fault tolerance method is called "recovery blocks" and conducting "acceptance tests" on the system. At various phases in the program, the entire program state is recorded. If any one of the acceptance tests fails, the appropriate state is restored, essentially pushing the system time back to that before the error occurred. If the same state is required to be restored multiple times, an alternate error handling routine is executed. The alternate method may be anything from running a substitute method, ignoring the process that generated the error, if possible, or requesting human intervention to correct a procedure [Kreutzfeld, 1997]. While there is enough slack in the system to perform the acceptance tests, there is neither enough memory nor extra processing time to keep track of multiple program states. But we can use acceptance tests in conjunction with other methods.

4. Error Classification

Keeping in mind the hardware restrictions incurred with the PANSAT system, the first step in building the fault tolerance plan for the satellite is to determine what kinds of problems could occur. It is impossible to identify every possible error that could occur, since the circumstances under which these errors occur is nearly infinite. However, by creating classes of errors, any fault that might arise should fall into one of the classifications and be properly handled, given the assumption that the error classes are correctly designed [Kopetz, 1979].

Only after the types of errors have been classified are the tests which identify these errors developed. These tests, the *acceptance tests* mentioned above, are created for each and every subsystem in the program which is being made fault tolerant. The approach is to identify each subsystem and match up all the possible error classes that could occur in the subsystem. Every possible symptom for each error class is then identified and a test to detect these symptoms is developed. These tests are the acceptance tests for that subsystem. This process must be repeated for each subsystem in the program since each has independent behavior and an error must be isolated as much as possible to aid in correction. Acceptance tests are discussed in more detail after completing the description of the error classes [Pradhan, 1986].

For the PANSAT project, three error classes are evident. The first type is errors that occur outside of the scope of the User Services software, system errors. This includes problems with uploaded software, including the operating system kernel, and system-wide problems. System-wide problems cannot be identified with any single program, but affect all the software elements. The second kind of errors are problems that could occur within the User Services program itself, program errors. This includes design as well as incurred errors. The final error type is data errors. These errors are not execution errors, like the other two types, but problems that occur with the information utilized or stored by the User Services program. Figure 2 lists the basic error subtypes specified for the satellite.

System Errors	**Program Errors**	**Data Errors**
Complete system failure Operating system failure Operating system livelock Watchdog program failure User Services terminates User Services livelock	Logic error in module Executable modification Program evaluation system failure	Dump of all files Corrupt file System log dump Data structure integrity violation

Figure 2: Software Fault-Tolerance Error Classes

While each error class is distinct, the schemes for handling them often overlap. Thus, before error classes are discussed, the overall scheme for fault detection must be defined. The main method developed for PANSAT is the Watchdog system, which relates to several of the error classes and often can comprise the entire solution to the problem. The sole function of Watchdog is to periodically send all operating threads a query on their status. If no reply or a bad reply is received, Watchdog takes appropriate action. According to initial estimations, performing the query cycle once every ten minutes should ensure that the system does not get bogged down performing tests, yet still permit catching important errors before they do damage.

The Watchdog program is unique in its functionality of regulating faulty components in the PANSAT system. Watchdog interacts with the other elements in the system to identify and error the error. On more systems with more available resources, a typical monitoring program would function as an arbitrator in the voting method, previous described. The monitor would evaluate the multiple answers provided by the system and attempt to choose the correct one. An example of this common monitoring implementation is the Space Shuttle. On the Space Shuttle, one computer monitors the results of two banks of two computers. If the two banks agree, the answer is propagated; if the answers disagree, intervention is required and some computers may be shut down [Lee and Anderson, 1990].

Unfortunately, PANSAT does not have the resource capability to achieve multiple independent answers. Instead, the Watchdog function oversees activity to isolate an error, then eliminate it. Only then can the correct answer be determined. Thus, Watchdog′s critical function is to repair the system rather than masking over erroneous results. This is the only way to overcome faulty behavior in a resource-limited environment.

5. System Errors

The first error class is one of the most disastrous. A complete system crash means the entire system resets or starts over. This could happen as the result of a power failure, which is hopefully a remote possibility. It is still a possibility, however, in that it could happen as the result of a traumatic experience, such as the satellite being hit by a foreign object. Unfortunately, once the system is reset, all volatile memory is reset as well. This means that the operating system and working threads, including User Services, are erased from memory. They need to be reloaded from the ground station. The only way that the ground station will become aware that the software needs to reloaded, or even fixed, is by the lack of responsiveness from the satellite. Once normal operations begin, all communications with PANSAT will be done directly with the User Services software, implicitly handled by the operating system. When User Services fails to respond to the ground station, the ground station can make a connection to the BIOS level of the satellite′s hardware. The BIOS level connection is restricted to the NPS ground station only. This is the connection mode that the SCOS and User Services modules are uploaded to the satellite. From this level, the ground station can determine that the operating system and threads are no longer on the system and need to be reloaded.

Once the software is reloaded, operations can resume. However, all the email and binary files that were previously sent to PANSAT would be lost, since they will be located in the volatile storage banks. The telemetry, system settings and log data should not be lost as they are stored in the non-volatile flash memory. Thus, when the new file system program starts up, it can assume the four-megabyte storage banks are blank and create a new file structure in that memory block. In the case of the flash memory, however, the file system needs to check these banks for an existing file structure. If a proper file data structure exists, the system should not create a new file structure, but rather incorporate the files that already exist. Once the file system re-accesses these files, the ground station can download them and view them. Inside these telemetry and log files may be an indication of the reason of the failure. While a random event, such as being struck by a foreign object, may not be avoidable, if an internal reason for the failure exists, the ground station may be able to implement a solution to prevent the failure from repeating. Unfortunately, each system crash must be handled on a case-by-case basis and involves active intervention on the part of the ground station.

Two more types of disastrous errors are the operating-system-failure and operating-system-livelock error classes. In terms of a fault-tolerance plan, both error classes are handled identically. Livelock is the condition where the program is performing an action, but no real progress is being made. An example of this would be if the program surreptitiously jumps into an infinite loop with no means to exit. The program is working; there is just no means of accessing it. In both error cases, the symptoms will resemble a system crash to the ground station: No connection to User Services will be possible. This is due to all communications with the satellite being directed or channeled by the operating system.

When the ground station enters the BIOS level connection, however, the operator will be able to tell that SCOS and the threaded programs are still active. Since the operating system has been extensively tested and used, such an occurrence generally should not be attributed to a bug in the program. More than likely, an element of the operating system has been altered by a space anomaly. In order to return PANSAT to working order in the quickest time, SCOS and the operating threads should be terminated at the BIOS level, then reloaded from the ground station. Just as in the system crash scenario, the flash memory will still retain all its data. However, in this case, the four-megabyte storage banks will also not have lost any data. The file system should, instead of creating a new file structure, use the preexisting structure. Once the system has become operational, there should be no loss of data. The net loss from these errors would be just the operational time lost before the error was found. Once again, however, this error must be handled manually by the ground station. The remaining error classes are less traumatic and have solutions that can be, at least in part, handled by the system itself.

The next error class that might occur would be an error in the Watchdog program itself. As mentioned earlier, the Watchdog program periodically queries the operating threads. In order to determine when it has errors, the User Services also serves as the Watchdog′s watchdog. In particular, if the User Services does not get queried for a period of fifteen minutes (a grace period in addition to the standard polling cycle time), a software flag is raised. The Watchdog program has either unexpectedly terminated or jumped into its own livelock condition. In either case, the User Services program continues communications operations as normal. The first error solving that User Services does, however, is to send a stream to the operating system to restart the Watchdog, which equates to setting the Watchdog′s program counter to zero. If fifteen minutes after this stream has been sent to SCOS and no query has been received from the Watchdog program, the User Services logs the event that the Watchdog is not performing. Since Watchdog′s performance is not critical to operations, no other programs need to cease. However, during their next connection, the ground station will be notified that Watchdog needs to be terminated, if it is still resident in memory, and reloaded. With the program reloaded, operations should continue as normal. In this case, the system tries to handle the error by restarting the Watchdog program. Only if the error is more complicated than Watchdog can handle does the ground station need to intervene.

Just as the Watchdog program may crash or go into livelock, the User Services program may do so as well. The Watchdog program detects either of these two error classes when the User Services program fails to respond to the query message. If the operating system reports back to Watchdog that the thread it is trying to send the query to does not exist, the User Services in this case, Watchdog determines the program has terminated and is no longer in memory. The Watchdog then logs the error. Next, Watchdog claims the User Services callsign. By doing this, when the ground station tries to connect into the User Services (the communications functionality), they will get Watchdog instead, which will simply provide an error message informing that the User Services needs to be re-uploaded. The benefit of doing this is to lessen the time the ground station takes in finding the error. Otherwise, the ground station would have to fail trying to connect to User Services, make a BIOS level connection, then determine the problem from there. This way provides the error detection up front.

If, however, the operating system believes that the User Services program is still in memory and just not responding to messages, the User Services thread is probably in livelock. The next step is for the Watchdog program to request the operating system to reset User Services, as described above. If after a reset, the User Services module does not respond to the next round of queries, Watchdog logs the fact, then requests termination of User Services from the operating system. This allows the Watchdog to perform the above paragraph′s actions just as if the program had already been terminated. Just as in the Watchdog error class, simple errors may be handled by the system before the ground station is required to intervene to correct a problem.

That completes the error classes in the system errors group. This fault-tolerance plan centers on the User Services module, since it is the cornerstone of PANSAT′s operations. However, just like when the User Services module crashes, Watchdog can detect when any of the other threads that might be on the system fail or enter a livelock state. Watchdog handles these errors the same way it handles User Services failing. The exception, however, is that the other threads do not have their own callsigns to connect to. Therefore, the ground station will have to check the system log entries to find an error in one of these other modules. This is not a problem however, since the ground station will be downloading the log record during every pass of the satellite. Since the User Services module should be working perfectly if the error is in another module, getting and reviewing the log entries would be performed normally.

6. Program Errors

Other errors that might occur are program errors. While the system errors deal with a complete program, or even the entire PANSAT system, failing, program errors arise within the program itself during execution. These problems however, do not cause the entire program to fail. Rather, a single component of the program fails or produces a wrong result while the rest of the program works as expected. Without fault tolerance, the program would normally continue executing, using the wrong result obtained from the faulty component. Fault-tolerance techniques are used so that when a wrong result is determined, it is evident and can be invalidated or corrected. As stated before, the User Services program is the focus of this determination.

The first error class in this type, logic or programming errors, can be found in any program. Ideally, all logic errors would be identified in the testing phase of program development. Realistically, however, some bugs will be missed and not found until actual operational use reveals them. Unfortunately, the system will not be able to tell the difference between this type of error class and the error of program modification by space anomaly. Program modifications have the consequence of a module that was once providing correct results now providing faulty results. If the program modification was so severe that it locks the whole program or livelocks when the procedure is invoked, the error class is no longer within the program error types, but becomes one of the system error types. Although both the logic error and program modification error have different causes, the system only sees the same incorrect results.

These wrong results are generally found in response to a Watchdog query message. When the User Services program receives a query message, before it replies, it conducts a series of tests to ensure that its subsystems are performing normally. These tests are the acceptance tests mentioned previously and defined later.
Once a bad subsystem has been identified by the acceptance tests, User Services notifies the Watchdog program, which logs the problem. The ground station can then review the problem and upload the solution. Of course, uploading the fix most likely involves gracefully terminating the User Services program and sending up a fixed version of the code. By performing the graceful termination, the ground station will be able to preserve all the data on the system. Thus none of the communications files will be lost, minimizing the impact of the error. If the program errors caused incorrect data to be stored, however, some of the data may be permanently lost. Meanwhile, if the problem was just a program modification, the code would not need to be modified from the original form. A logic error would require an updated version of the software to be created. The ground station is responsible for determining which of the two error classes caused the problem. This is done by viewing the log data and comparing the results with backups of the satellite software maintained at the ground station.

While the solution to the error is being handled by the ground station, the satellite may be able to keep performing, albeit in a diminished mode. The User Services software will maintain a table of subsystems which correspond to particular services provided by the program. When a subsystem is determined to be faulty, User Services will simply not use or offer the corresponding service obtained from the table. If the denied service is mission-critical, such as ability to put data into a transmittable packet, the program will terminate. If the service is not mission-critical, it will notify users who log into the satellite that a particular feature is temporarily unavailable, for instance, retrieving the current telemetry might be removed from the menu list if the telemetry subsystem failed. The goal of handling this error is to provide as much communications functionality as possible until the problem is rectified.

A special case of the program modification error is the program evaluation subsystem of User Services itself suffering from a space anomaly. The Watchdog program checks the response sent by the User Services program. If all of a sudden User Services reports errors in every single one of its subsystems, the error most likely lies inside the subsystem conducting the testing, not the other parts. In this case, Watchdog logs the suspected error and keeps operations as usual. The Watchdog program will continue to query User Services, just to determine if an unexpected termination or livelock occurs. The actual response sent back the User Services will be ignored, since the testing system would be expected to be flawed.

In all three of the error classes in the program error type, the ground station is required to intervene to correct the problem. This is due to the programs being held in the volatile working RAM. No copy of the code exists on the satellite to determine which bits might have been flipped. However the key to this fault-tolerance technique is that service interruption is kept to a minimum. Early error detection and notification of the ground station, coupled with temporary work-arounds by the system software itself, allows continued services to be provided to the end user until those fixes can be implemented.

7. Data Errors

The final of the three error class types is data errors. For the most part, the previous errors described were dealing with the executable part of the program being incorrect or altered. Data errors, on the other hand, involve the programs working perfectly. The data that is being worked on, however, has been damaged in some way. This means that most of these errors are detected via the acceptance tests.

The data error with the largest scope is the event of all the files in the storage banks being dumped or erased. When this occurs, and the User Services program is still executing, a system reset is not the cause of the storage dump. In this special case, the contents of memory may still be intact, only the file allocation table (FAT) may have been corrupted. The system will first attempt to reconstruct the FAT from what information can be found in the memory location where the FAT is normally kept. Aiding in the chances of rebuilding the FAT is the fact that the FAT is created in duplicate. Thus, out of two corrupted FATs, the possibility exists for a complete and correct table to be created. Also helping with the recovery of the FAT is the fact the files are created by User Services in a very methodical way. Only two groups of files exist in mass storage, email files and binary files. They all use the name scheme of m#### and f####, where # represents a digit from 0 to 9. Conducting a search through the memory banks may reveal the files are still present and reconstructable. Knowing this and the data structure format of the FAT, there may be enough information obtained to create new FATs from the files left in memory.

If attempting to recreate the FAT fails, the files are unrecoverable. PANSAT will have to establish a new file structure in the four-megabyte storage area and start from scratch. For one week after such an incident, any user who communicates with the satellite will receive a warning message that all data was lost on the date that it occurred. That way a user can re-upload an email or file as necessary. In either case, the problem is logged so the ground station can review it. While the ground station cannot restore the files, if the cause of the problem was more than an isolated incident, the ground station may be able to determine the error source and fix it.

A less severe data error is the corruption or loss of a single file in storage. Hopefully, since the storage area is using mirrored files, one of the two storage banks contains a correct version of the file. However, corruption of a file can still occur in both memory banks. Typically, a corrupted file only has a section of the file lost. When a file is found to be corrupted, User Services will rebuild the file with all the information that could be recovered. A marker will be put in the place where the data was lost. For instance, in an email message, the place with the missing text would be replaced with the “[text missing]” caveat. Additionally, the originator of the message will be sent an administrative message indicating which message was damaged and indicating a retransmission may be necessary. Of course, the originator would get this warning in the next communication with PANSAT. If no discernable data is retrievable from the corrupted file, specifically the originator of the file, the file is simply deleted from the system. Just as with the file system dump error above, the ground station is notified of the error for evaluation, but will be unable to do anything more in the recovery of that particular corrupted file.

While the Watchdog program would determine a problem in the program that managed the logging system, it would be unable to know if the log file itself had unexpectedly been erased. This is the next type of data error. Since the log file will be maintained in the non-volatile flash memory, erasing it would most likely be the result of an inappropriate software action rather than a hardware glitch. When the system-logging program responds to the query from the Watchdog program, it checks to ensure the contents of the log file are intact. If not, the Watchdog program is notified. Unfortunately, with the logging system temporarily non-functional, the Watchdog system is unable to use its primary method of notifying the ground station of errors. In this special case, the Watchdog program sends the User Services a special message informing it that the logging system has an error. In its next communication with PANSAT, the User Services will notify the ground station of the logging problem.

Note that this procedure is used not only for a problem in the log file, but if the system logging program has a fault as well. The ground station will have to determine the nature of the error. If the logging program is faulty, it will be reloaded. If the log file is accidentally deleted, the ground station must determine which program deleted the log file. This will have to be done via a recreation of the satellite’s environment at the ground station. As much information about the current state of the satellite will have to be provided by User Services in order to aid in the recreation process. These errors are a special case in that the normal means of error notification, and thus error rectifying, is removed from the system. Thus, even though operations continue for the satellite, the ability to handle and fix any other error that might happen is drastically reduced until the logging system error is fixed.

The final error class identified for the PANSAT project is the corruption of a program data structure used within the User Services program. The User Services program functions as an automata state machine, acting on data from an end user based on what state the system is in for that particular user. Thus, if a data structure becomes corrupted, a user connection may act unpredictably. Fortunately, the corrupting of a data structure will most likely affect only a single user; the other connections will be unaffected. Once a data structure is determined to be corrupt, there is a good chance that it can be rebuilt. The operating system provides much of the state information as it delivers a packet of data received from the user to the User Services program. Thus, using that information, much of data maintained about the connection can be determined. If, however, a complete data structure cannot be rebuilt from the data provided, the User Services needs to interrupt the connection by sending the user a series of interrogative messages. After these messages are sent, the program should put the connection back into the default state, and should print out the main menu of choices for the user to pick. The user would then be responsible for salvaging the session and retrying to complete the work that was ongoing at the time of the fault.

As with all other data errors, the corrupt data structure error will be logged and as much data will be recovered as possible. The ground station will be notified of the error, but will not be able to fix that particular instance. Using the data, however, a solution to the problem that created the data loss in the first place may be determined, hopefully eliminating future errors of the same kind.

This concludes the classification and handling of errors that PANSAT might see. Although each specific error that might occur is not detailed in this plan, most likely any error that arises will fall into one of the error classes and be handled appropriately. No matter what, PANSAT always has the BIOS level connection which can be used to reset the system to start over. If for some reason this connection is unable to be made, the satellite has suffered a major catastrophe and, for all practical purposes, is dead. While there can be no mechanism to handle this type of an error, the likelihood of this happening is small.

8. Acceptance Tests

When discussing the program and data errors above, it was taken for granted that these errors would be detected by the acceptance tests mentioned. As mentioned previously, without a complete idea of the errors to create the tests for, the acceptance tests can not be fully designed. Once the error class plan is formulated, however, it is possible to develop the precise tests to fulfill the plan. As implied above, the acceptance tests should test all of the subsystems and data organizations to determine their reliability and integrity [Pradhan, 1986]. Figure 3 shows the four types of acceptance tests the PANSAT User Services program will perform to conduct a self-evaluation. Each one of the classifications is discussed in the following the table.

Acceptance Test Classifications
Satisfaction of requirements
Accountability tests
Reasonableness tests
Computer run-time checks

Figure 3: Fault-tolerance Acceptance Test Classifications

The first acceptance test type is satisfaction of requirements. This means the subsystem provides the expected results. Naturally, each subsystem will have a different method for checking its results. A standard means for procedures that take data in, process it, and return an answer is to use table comparison. Obviously, it is impossible and impractical to maintain a list of all legitimate answers for any given input. Rather, when the Watchdog program′s query message is received, the User Services testing procedure refers to a table of predefined tests for each subsystem. There should be only one or two tests per subsystem or the program could get bogged down performing all the tests. The tests, however, should be a good representation of the type of calculations required by the system. Included in the table is the correct answer the subsystem should return for the test. Any variance between the table answer and the subsystem provided result would be a logic or program modification error, as described above, and reported to the Watchdog system.

This type of comparison test is used on the subsystem which performs the packetizing of data. All data sent from the satellite must be in AX.25 packet format. By providing preselected raw data to the procedure that forms these packets, then comparing the format of the packet to what is expected, a determination of a flaw arising in the procedure is be achieved. Additionally, the comparison test is performed on the data compression/decompression algorithm. A set of compressed data is sent to the procedure to see if the correct data is extracted, and vice-a-versa. Finally, the position-determination routine is tested with this technique. The routine takes an initial position and a change in time since the position and produces the current position. The comparison test table contains all three elements to determine the position routine accuracy.

If the subsystem is not answer-based, but rather performance-based, the User Services could check that the action performed was actually the expected result. While the answer-based checking would be done only in response of a Watchdog query message, the performance-based procedures would be checked after the completion of every action.

On PANSAT, the system that saves a file into the storage memory banks is be checked by determining whether the file name and size of the newly created object is the same as that provided to the file system. Once again, a mismatch would be classified as either logic error or program modification error. Additionally, when an entry is sent to the event logging subsystem, the event log file is checked to ensure the entry was appended to the end of the file. The defragmentation module also uses a performance-based satisfaction test. After a file has been defragmented, the file data is looked up in the FAT to ensure that it is now taking up on contiguous block in storage.

The second kind of acceptance tests are accounting tests. That is, User Services keeps a tally of certain information about the rest of the system. When a particular subsystem is queried about its tally, if it does not match User Services′s version, an error is declared.

Accounting tests are used to compare a file counter held by User Services with a directory count provided by the file system. A variation in the number could indicate a corrupted FAT, which is handled as a dump of the file system error. In addition to that procedure, the number of packets returned from the packetizing system for a certain sized block of data should be a predetermined amount. A variation would be a logic or program modification error. Next, a timestamp is placed on all recorded activities of the User Services program, such as saved files, transmitted packets, or positional estimates. If the timestamps are not ordered, the system′s clock interface system may be flawed. Finally, connection records are checked to ensure that the number of in use records matches a counter which indicates the number of ongoing connections. If the numbers differ, a connection has been lost. The handler for a data structure integrity violation is invoked.

The next acceptance test type is called reasonable tests. This name says it all: whether the system provides a reasonable result. This is the most common acceptance test and is used with almost every subsystem in the User Services package.

For instance, this test is used when telemetry values are obtained. Each value procured has a possible value range located in a table. If the value is outside of the range, or the value has changed within the range, but at a rate that is not physically possible, then the means of getting the values or the sensor providing the value may be faulty. Manifestly, if a user requests an operation that does not make sense, the operation should be rejected and a warning sent back to the user. If data returned from a function is not the correct size, it is faulty. This includes the size of packet received from SCOS to the length of a datastream connecting processes. A position returned by the dead-reckoning procedure must be within a range of possible positions on the satellite′s orbit. A position of the North Pole would be flagged as faulty since PANSAT′s orbit does not come close to the North Pole. Finally, settings commands received from the ground station must make sense. That is, the settings must correspond to feasible operations. For example, performing an auto-purge of the storage banks every minute is not a practicable operation.

The last kind of acceptance tests are labeled computer run-time checks. These are handling typical run-time errors that usually cause a program to terminate. Also called exception handling, these functions catch divide by zero, overflow, underflow, and other errors by abstracting their operation into a single safe procedure. By trapping their execution, the program will continue to execute, and the subsystem that requested the illegal operation can be determined. This is implemented by placing all the exception routines into a single checked module. Once the module is proven safe from crashing, any element that uses the module is safe from crashing due to checked run-time error. It is important to keep the program from terminating since, as mentioned before, once the program terminates, it is removed from memory. Removal would prohibit determination of the offending subsystem and make harder the fixing of the error.

Another run-time check is for each procedure to check the program stack upon being called. If the stack is not formatted correctly, the procedure may have been illegally jumped to. A correct stack format would be indicated by the proper type and number of parameters being placed on the stack. It may be impossible to determine where to resume execution from when returning from an improperly called function, so a safety point of reference, the main menu in this case, may have to be jumped to in order to continue execution.

These acceptance tests are very specific to the individual subsystem that they are evaluating or protecting. The basic concepts presented here form a comprehensive coverage for many systems, however. When developing a list of tests, every subsystem should be evaluated. In general this type of fault-tolerance plan increases the overall work load by 25 percent, an acceptable amount.

9. Conclusions

None of the technologies applied in this fault-tolerance plan are difficult in and of themselves. However their application in conjunction provides integrated comprehensive coverage, which dramatically increases the reliability of PANSAT. As mentioned at the beginning, most fault-tolerance research has been working with distributed systems, working with a large pool of hardware resources [Kreutzfeld, 1997]. Unfortunately, the particulars of space, especially in this project, preclude the use of most of those features. The methods instituted on PANSAT are tried and true fault-tolerance methods, only they have not been previously linked together to form a consolidated fault protection for a system. Using this innovative plan, the satellite should be able to operate and provide a level of reliability expected by the military for a communications satellite.

Limited resource fault tolerance can be successfully implemented. While it may not provide the same level of self-correction that is expected of a large resource system, the errors are still trapped and handled by the limited system, with a little bit more of an active role by the ground station.

10. References

Avizienis, Algirdas, *Toward Systematic Design of Fault-Tolerant Systems*, IEEE Computer Magazine, April 1997.

Bible, Steven R., and Daniel Sakoda, Petite Amateur Navy Satellite, M.S. thesis, NPS, 1998.

Kopetz, H., *Software Reliability*, Springer-Verlag, 1979.

Kreutzfeld, Robert J., and Neese, Richard E., *Methodology for Cost-Effective Software Fault Tolerance for Mission-Critical Systems*, IEEE AES Systems Magazine, September 1997.

Lee, P.A., and Anderson, T., *Fault Tolerance Principles and Practice*, Springer-Verlag, 1990.

Pradhan, D.K., ed., *Fault-Tolerant Computers: Theory and Techniques*, Volume II, Prentice-Hall, 1986.

Somani, Arun K., and Vaidya, Nitin H., *Understanding Fault Tolerance and Reliability*, IEEE Computer Magazine, April 1997.