**CS630: Modern Operating Systems**

**Individual Project 5: “iSpace”**

**Ajitpal Grewal**

**4/21/2024**

[1. Project Outline 3](#_Toc164695267)

[2. OS Processor and Core 5](#_Toc164695268)

[3. Scheduling Algorithms 8](#_Toc164695269)

[4. OS Concurrency Mechanism 13](#_Toc164695270)

[5. OS Security Risks and Mitigation Strategy 31](#_Toc164695271)

[6. Future Considerations Using Emerging Technology 47](#_Toc164695272)

7. References………………………………………………………………………………………………..54

## Project Outline

* In the early days of space exploration, the technology available made use of single-processor and single-core operating systems in cases such as batch processing. The limited resources and the processing of information sequentially caused instances of bottlenecks (Stallings, 2017). With the advancement of technology, *iSpace* aims to be the global leader of everything involving space exploration. This will include using satellite technology to observe the depths of the universe, and aerospace technology to send rovers and astronauts to far reaches of the galaxy.
* On the ground, iSpace currently utilizes the IBM System/360 as its primary operating system configuration. In this specific single-core architecture, one CPU executes processes and performs calculations. The CPU includes components such as the arithmetic logic unit, the control unit, and registers. The ALU carries out mathematical operations using carry signal propagation (Ying, 2020). The control unit oversees obtaining instructions and executing functions. Registers can be used to store data (Stallings, 2017).
* The IBM System/360 is the focal point in mission planning, data analysis and telemetry of the spacecraft. It also utilizes Random Access Memory to store program data and access it using memory addresses (Jenkins, 2001).
* The use of the single-processing, single-cored IBM System/360 is inefficient in many cases and forces the enterprise to create specialized computers for each individual mission, catering to mission needs. Each clock cycle does not process enough information fast enough and the system bus must be quicker. Carrying out one process at a time simply takes far too long if the company wants to advance in a competitive environment.
* The IBM System/360 utilizes redundancy and fault detection tactics in the form of hardware. Examples are redundant power supplies (Jenkins, 2001).
* Much of the data is stored in a few main facilities on tape drives (Campbell-Kelly, 2015). The fact that some information is not as easily accessible as other information can sometimes lead to latency issues. The servers themselves are single-core systems.
* Tasks having a heavy workload and only a single-core operating system may not be feasible.
* The enterprise uses custom Real-Time Operating Systems on spacecraft with software routines to ensure swift decision making. An example is the Apollo Guidance Computer which controlled navigation and guidance. An outdated language is tailored specifically for each process within the IBM System/360 (Jenkins, 2001). The real time operating systems themselves take resources to create and are specialized depending on the operation. One system could control the vehicle direction while another controls data processing. Those single-core processors that are built into the aerospace vehicles themselves have low processing power. The ability to scale is also limited with increased workload.
* Each software program is limited to carrying out a specialized function with little to no modification ability. This is due to factors such as the necessary safety functions of the program or the actual code used to create the program and compatibility with hardware (Jenkins, 2001).

## OS Processor and Core

* The introduction of the Linux operating system will give the company a vast amount of flexibility, allowing it to allocate resources properly and improve performance. The availability of Linux’s source code allows iSpace to be intricate during customization. The Linux kernel can be manipulated to cater to the needs of each mission. When the Linux kernel makes a context switch from one process to another, it checks if the current address process is the same as that of the next process. If they are sharing the same address space, a context switch is a jump from one location of code to another location of code (Stallings, 2017).
* This flexible quality of the Linux kernel may reduce overhead simply due to the reduction of steps involved in switching. This would lead to faster performance and less work. It would also allow for even distribution of workload. The open-source nature would also allow compatibility of diverse features, saving time and manpower on catering specific tasks or debugging.
* The company aims to upgrade the single-core and single-processing units to multiple-core and multi-processing systems in their vehicles and ground control units. The upgrade to Linux will bring advancements such as parallel processing and multitasking. More communication between processors will allow for a purposeful distribution of resources, and no one processor will be overused. This will inevitably lead to the conservation of resources and efficiency of outcomes as in the case of throughput. The Linux kernel will also allow for heightened scalability because of quick context switches (Stallings, 2017). This quality makes it suitable for an aerospace environment, allowing the OS to manage large numbers of processes running concurrently. Swift context switching reduces idle time across all processors, making load management more cohesive. As more tasks are confronted, the OS can manage the distribution of information without being overwhelmed, and allocation of task completion is simple.
* The safety inside each vehicle would increase due to the multiple failsafe measures in cases of fault tolerance and redundancy. Not only will time be saved, but security measures in cases where fault tolerance is crucial will be strengthened. The original systems used processes where the entire operation could fail due to a single point error. The reliability of systems involved in space exploration is nothing short of mandatory.
* Linux uses software approaches to offer redundancy. Examples include RAID 1 or RAID 5, which provide fault tolerance. Cluster configuration in Linux can allow for leniency of server failure. Cluster configurations are often modified with data replication to increase reliability (Stallings, 2017). The backup of data across multiple servers may be crucial in the case of an unforeseen problem.
* With these advancements, instances of deadlock and lag would be reduced (Stallings, 2017). In the case of space travel, the speed of processing and communication between sectors cannot be underestimated.
* iSpace would implement “SpaceCloud” to transfer data between large distances. This will allow for proper communication and quick use of systems. In the case of long-distance data transfer, placement of multi-core servers across strategic geographic locations is crucial to efficiently move information in a speedy manner. This, in conjunction with cloud communication, will allow for quick decision making and ensure proper function. The Linux OS in a virtual environment utilizes a hypervisor that abstracts the physical hardware of the actual machine. The hypervisor will gather resources such as memory and allocate information accordingly across all necessary virtual machines. This allows for a single machine to equivocate the workload and effectiveness of multiple machines (Stallings, 2017).
* In contrast to IBM System/360, Linux can run on a much wider range of hardware due to its architecture compatibility. The IBM System/360 is limited to specific hardware (Jenkins, 2001). The Linux OS can support multiple hypervisors. The open-source nature of Linux has allowed for a broad range of hardware compatibility as well (Stallings, 2017).
* The modernization of computer language has allowed for flexibility of software-hardware compatibility, limiting the use of Assembly Language.
* The enterprise needs increased processing power, scalability of workload in real time and more modernized flexibility of function and compatibility of computer languages with software and hardware.
* The company will identify compatible multi-core and multi-processor hardware that is best suited for Linux in high altitude environments. The architecture of ARM and x86 processors are optimal for energy efficiency. The installation of multi-core processors using proper guidelines will reduce power consumption and decrease calculation time in cases such as flight path correction. The cost of IBM System/360 configurations can cost millions of dollars due to their need to be heavily customized (Jenkins, 2001). The concrete nature of this system makes it a less feasible machine. The typical Linux operating system ranges in the tens of thousands, making it the optimal choice.
* The upgraded systems will be tested in simulated environments that best represent real time decision making. Linux software can simulate real environments such as in the case of aerospace. Pilots can be trained, and weather can be modeled (Lockney, n.d).
* Upon testing, the gradual integration of upgraded systems and training of all necessary personnel will be monitored, acquiring feedback from all relevant parties. Adherence to guidelines laid out by the Federal Aviation Administration will be strictly enforced. With continued monitoring, negative feedback will be closely recorded and fixed in a timely manner.
* Continued maintenance is crucial to ensure operator safety. Constant debugging, patching and 24/7 technical support will continue to see advancements.

## Scheduling Algorithms

* iSpace currently utilizes non-virtual machine environments in its spacecraft, rovers, and satellites. Each entity would be embedded with custom Linux-based Real-Time Operating Systems specializing in functions that adhere to their mission objectives. An OS with a non-virtual machine environment is suitable for aerospace missions due to its stability and responsiveness. The OS interacts directly with the hardware such as input/output devices. The spacecraft will have embedded Linux-based Real-Time Operating Systems and the ground servers will be distributed across strategic locations. The distribution of servers will allow for fault tolerance and redundancy to ensure no data is lost (Stallings, 2017).
* The RTOSs aboard rovers will be embedded systems and their qualities defined by their mission goals. These embedded systems will communicate with a satellite via radio frequencies. This collected data will then be communicated to ground servers upon reentry to Earth’s communication proximity also via radio frequencies. In ground servers, the data is saved on hard disk drives. File systems are organized methods of how data is stored and utilized (Stallings, 2017).
* Direct hardware access of non-virtual machine environments in aerospace conditions allow for increased reliability and fault tolerance. Direct access to memory modules and storage devices can increase the speed of real-time operations. By reducing the overhead of a virtualized environment in spacecraft, the systems can adhere to stringent timing deadlines. With each physical machine running its own tailored operating system, the machines can experience the full advantage of utilizing their own CPUs and hard drives. This availability will increase the chances of mission success.
* Custom embedded operating systems such as these can switch between tasks quickly with their real-time scheduling policies. For example, the OS ensures each step towards course correction is taken to avoid collision in a spacecraft. The dispatcher module prioritizing the next task is built into the system itself, which also allows for quick decision making. The OS responds to external interrupts, such as sensor detection, swiftly. The OS also attempts to keep these interrupts as short as possible. A typical response time is less than 10 microseconds (Stallings, 2017). Although in this type of environment, that number is likely to vary.
* In this environment, the operating system manages memory by fixing it into sizes. This prevents code from being corrupted. The OS can store data from sensor readings rapidly by using special sequential files. The kernel keeps track of time constraints by setting alarms. It utilizes scheduling methods such as the EDF and RR algorithms for urgency and can control the execution/suspension of functions (Stallings, 2017).
* Real-time processing in conjunction with effective scheduling algorithms can lead to adequate response time. Some of these algorithms include the Earliest Deadline First (EDF) and Round Robin (RR) algorithms. In the spacecraft operating system, the EDF algorithm associates urgent navigation tasks with higher priority deadlines. To ensure safety and avoid collision, each task must be done on time and lower-priority processes are preempted. This preemptive scheduler allows for the minimization of the number of tasks that miss their deadlines and maximizes system efficiency (Stallings, 2017).
* If there are multiple tasks with the same deadline, the scheduler will further analyze the situation and gather more information to decide which process is to be executed first. This decision is of the utmost importance. In the example of avoiding space debris, the EDF algorithm can assign task deadlines based on assumed trajectory and time until impact. The enforcing of deadlines and making sure tasks are executed within an allotted time is crucial in real-time course correction. This allows the OS to execute the next urgent task and keep the flow of processes moving (Stallings, 2017). Starvation must be avoided at all costs to ensure safety. The dynamic nature of the EDF algorithm is dependent on the expected deadline of the tasks themselves.
* This algorithm involves the interruption of long, low priority tasks. Suspending a slow process and allowing a shorter process to execute is an important characteristic of this scheduler. This allows the system to have proper utilization of resources. The higher priority tasks that get moved to the front of the queue must have full access to their required resources. This characteristic also allows the EDF algorithm to scale with additional processes being loaded into the queue.
* The Earliest Deadline First algorithm’s dynamic and quick switching qualities make it a strong choice for this environment. It can account for things like changes in the environment accordingly. The attention to urgency of this algorithm allows for exponential scaling. If done properly, this algorithm can also increase throughput of the system by getting shorter processes done in a small window of time (Stallings, 2017).
* The EDF algorithm ensures there is little to no idle time. Each process is allocated resources as needed. This avoids instances of deadlock. Each resource is distributed amongst processes when necessary to complete tasks within their deadlines (Stallings, 2017).
* Another advantage the enterprise can see with this algorithm is its ability to schedule repairs based on their deadlines. The EDF scheduler can assist with scheduling tests and mission planning as well. In large scale operations such as this, each event must be treated with care to ensure safety. The use of the Earliest Deadline First algorithm can reduce the chance of missing critical steps along the way.
* A challenge that comes with the EDF algorithm is predicting the deadline of an unforeseen task. For example, sporadic movements of space debris may cause the deadline of a navigation task to be incalculable. Even if the debris is detected on time, it may be impossible to determine its priority. The mission could be jeopardized due to the inability of impact time determination.
* Resource allocation may be problematic if an unforeseen task does present itself. If the system is constantly required to prioritize navigation and collision avoidance techniques, then the tasks that are considered less important will starve. This will inevitably lead to the degradation of mission objectives. This is especially detrimental if resources such as fuel are limited.
* If a system equipped with this algorithm comes across multiple tasks competing for the same resources, deadlines may not be met. A missed deadline could mean the entire mission fails. If the system is forced to balance a high workload, constant switching between processes may be wasting time instead of conserving it.
* This algorithm may wrongfully prioritize an unimportant task based on their deadline times. For example, a non-catastrophic piece of debris may be close to impact but does not need prioritization of execution because there is another larger piece of debris that needs to be maneuvered around. There must be other factors the algorithm should be supported with besides solely the deadline. This can include the size of the object in question.
* Relying heavily on external sensor information can be detrimental for this algorithm. Since the EDF can make decisions based on signal readings, a wrong reading can lead to catastrophic events. The EDF would simply take the information given falsely by the sensor and execute the process it deems appropriate. An inaccurate sensor reading would then also lead to the depletion of resources.
* The Round Robin algorithm poses its own set of benefits and challenges with some similarities to the EDF approach. The RR approach implicitly allows all tasks to share resources fairly. While the EDF approach may cause some processes to starve, the RR algorithm would prevent any task from using up all the CPU time if it were confronted with a scenario such as collision avoidance. The next task in the queue would have to wait until the time quantum that has been put in place expires (Stallings, 2017). In the case of aerospace, these time quanta may be set to microseconds.
* If the quanta are set to microseconds, it allows the spacecraft to react quickly to environmental changes. In turn, this gives the spacecraft low-latency aspects. The quick and frequent preemptions made by the Round Robin scheduler keep the system running at a smooth and rapid pace (Stallings, 2017). As mentioned, quick reactions are key to mission success.
* The simple nature of the Round Robin algorithm makes it a practical approach in non-virtual aerospace environments. The easy implementation of the Round Robin scheduler can allow for fast fixes. In the case of a rover, predictable task times and executions are optimal. Suitable time quanta can be allotted to control the rover’s actions.
* This scheduler’s low complexity allows the system to keep in mind its objective without straying far from course. Since no task can occupy much of the CPU time, the overall goal will be maintained because processes can continue to run accordingly. This is especially important in scenarios such as data collection. The low overhead of this scheduler provides the system with reliability, functionality, and overall efficiency (Stallings, 2017).
* The Round Robin approach would allow the enterprise to dedicate time slots. This would allow for the transmission and reception of data sequentially. One instance of a time quantum could be dedicated to processing or sending the collected data, while another could be dedicated to receiving future commands. This would also allow for instances for a machine such as a rover to make sure it does not waste any resources.
* In a Round Robin algorithm, processes that seem to be hindering the overall objective are unable to occupy all the CPU time. This is especially important in instances of data collection where glitched processes may cause a rover to indefinitely recycle through those processes. Once again, the RR approach is seen improving fault tolerance. The approach allows a system to bypass components that are causing repetition. In some cases, those processes are infinitely looped, and the mission duration is strictly timed so it is mandatory that no one task causes the downfall of the entire operation.
* While the Round Robin algorithm is a strong scheduler in aerospace, it can be difficult to maintain in certain scenarios. In spacecraft, the importance of avoiding hazardous material may not be emphasized enough with the Round Robin approach. With no prioritization, an event that requires immediate attention must wait until the previous task has completed its allotted time slice with the CPU (Stallings, 2017). In some cases, it is beneficial to put aside the main mission objective to preserve operational integrity. For example, a rover must be able to traverse terrain before it can collect samples.
* In other cases, multiple long processes may also cause the Round Robin approach to be inefficient. If consecutive long tasks are unable to finish in their allotted time with the CPU, the constant preemption of tasks will cause overhead (Stallings, 2017). A rover collecting samples faced with several processes that do not finish within their time quanta will not be able to carry out its job. The Round Robin scheduler will keep preempting those processes and delay the objective.
* The Round Robin does not adhere to stringent, real-time deadlines such as the Earliest Deadline First scheduler. It is difficult to implement this algorithm in scenarios where swift course correction is required. One action in navigation may be done sooner than expected, but the next task must wait to be executed. This can be catastrophic in times where immediate maneuvers are required.
* The difference in duration of long and short processes can vary greatly in aerospace environments. This can make it difficult to predict an accurate time quantum for each process. The scalability of the Round Robin scheduler is minimal due to its equality features. Events tend to pile up if they are short, especially if the time quantum was not assessed accurately.

## OS Concurrency Mechanism

* Environment Discussion
* The harsh surroundings of aerospace force iSpace to implement suitable concurrency mechanisms. The importance of real-time precision and reliability cannot be overstated. The non-virtual Linux-based Real-Time Operating Systems (LRTOS) embedded in the functional units of the enterprise maintain strict and timely executions. The integration of software and hardware components in spacecraft, satellites, and rovers can be custom designed to control propulsion or allocate resources such as power. Because the software and hardware interact directly, the LRTOS can respond to external stimuli promptly. This minimizes the overhead that would be caused by using a virtual machine. By operating on physical hardware and avoiding abstraction, the machine can perform optimally in the aerospace environment (Stallings, 2017).
* The open-source Linux-based RTOS is flexible. It can operate a wide variety of applications and it is equipped with several developmental capabilities in its libraries. One such library that can be implemented is the Boost C++ library. The Boost C++ libraries are peer-reviewed, open-source libraries that extend the nature of normal C++ libraries. They can help with tasks and data calculation such as linear algebra or multithreading (Mukherjee, 2015).
* As previously discussed, the embedded operating system can handle scheduling based on priority and interrupts. The LRTOS can direct the passage of information between different software. Acting as the brain of the functional units, these embedded systems provide predictable outcomes because of their direct interfacing with instruments such as sensors.
* The Linux-embedded environment gives way to efficient kernel modification. The platform provider is not dependent on a single vendor for resources. The malleable nature of this OS gives the machine diversification qualities. Linux supports a variety of processor architectures and peripheral devices, making it suitable for various surroundings. As mentioned, the open-source feature makes this OS highly adaptable. Rapid innovations are seen constantly due to the collaboration of multiple users within the organization (Stallings, 2017). Each entity can modify its embedded system to cater to mission needs. The adaptability of the LRTOS to hardware allows for constant improvement in progress. Rather than starting from the beginning for each new modification, engineers can build from a pre-established workup. Utilizing existing code can save time and resources because they have no need to retrace steps.
* Navigating the aerospace environment is a profound challenge. Space exploration brings with it a set of unique variables that must be accounted for. This can include conceivable events such as a rover being obstructed by surrounding terrain or finding and installing applicable hardware that can withstand the physical environment of the mission objective. Engineers and data scientists apply meticulous calculation and trajectory prediction methods to carry out objectives. The trajectory of spacecraft is calculated using extremely precise metrics and propulsion analytics. These calculations are crucial for ensuring that spacecraft reach their intended destinations accurately and safely.
* Engineers do this by considering various factors such as spacecraft position, velocity, and planetary gravitational forces (Comins, 2017). The practicality of space exploration can be questioned due to the vulnerability of our systems. The LRTOS interacts with the hardware of spacecraft through drivers. This includes handling telemetry data, controlling actuators, and managing sensors. iSpace’s custom drivers promise real-time reliability during interaction with spacecraft hardware. The embedded RTOS of each of these systems must be carefully designed to fit mission standards and safety protocol.
* In navigating the aerospace environment, one must consider circumstances such as the performance of its propulsion systems. The Linux OS must accurately portray information to the actuators of thrusters. By interfacing with the actuators, the OS can direct the action of thrusters. The thrusters are provided with information such as which direction to move and thrust duration. Actuators are designed to manipulate a spacecraft’s position and to maneuver a spacecraft with the proper correction of velocity. Common spacecraft actuators include magnetic torquers and reaction wheels (Zhou, 2010).
* Asynchronous Message Passing - *How it Works & How it’s Used*
* In Asynchronous Message Passing, a process is not suspended upon the "Send" or "Receive" command. This primitive is nonblocking and uses events or remote procedure calls to communicate with the system (Stallings, 2017). Concurrency management mechanisms in the Linux-based RTOS allow the system to optimize performance by allocating resources timely and effectively. Concurrency mechanisms enable the operating system to differentiate and process data without risk of deadlock and starvation. These mechanisms reduce the instances of multiple threads or processes blocking one another from being executed (Stallings, 2017).
* When a process initiates a Send command in an asynchronous message passing system, the operating system quickly returns control to the process. This is because asynchronous message passing allows the sending process to continue its execution without waiting for the message to be transmitted or received by the recipient. As a result, the sending process can proceed with its tasks without waiting. If no copy of the message is made before transmission, any changes that are made to the message by the sending process afterward are made at the risk of the process. This is because the message may already be in process or received by the necessary component (Stallings, 2017).
* When the message has been transmitted or copied to a safe place for further transmission, the sending process is interrupted to be informed that the message buffer may be reused. Once the message is sent, the process that sent the message is paused and then resumes once it knows it can reuse that same message buffer. The Receive command is also issued by a process that then proceeds to run. When a message arrives, the process is interrupted, or it can do routine checks on the message. For example, in interrupt driven I/O, when the processor issues a command from a process that is nonblocking the processor continues to execute instructions from the process that issued the I/O command (Stallings, 2017).
* Asynchronous message passing begins with the creation of messages by a sender. These messages can contain data, commands, or signals for execution that are passed on to further processes. This message can also include the destination of the message. As mentioned, the Send command or system calls initiate transmission of a signal that is to be processed using a set of instructions within the system. System calls within the OS allow kernel interaction (Stallings, 2017). They can act as the “middleman” between user applications and the kernel. This interaction allows processes to perform I/O operations.
* There are several variations of message passing systems. Transmission of a message to another process is tagged with a destination (Karlsson, 2005). A process receives information by executing the receive primitive, indicating the source and the message.

The following is an example of a pair of primitives:

Send (destination, message)

Receive (source, message)

When the retrieval process begins, it implements the First-In-First-Out algorithm to decide which message is manipulated next. The recipient process can then digest the retrieved message according to specified requirements which can involve another round of message passing (Stallings, 2017).

* Asynchronous message passing (AMP) can be a strong candidate for event-driven operating systems. Such events can be user operations or external interrupts that require attention. In this case, the messages themselves can be interpreted as events meaning one message can trigger another. The idle time in blocking primitives can be detrimental to resource conservation, therefore asynchronous message passing and its nonblocking qualities allow I/O operations to operate concurrently. In AMP, processes can continue to execute while awaiting event/message arrival (Stallings, 2017).
* AMP may employ handlers as primary supervisors of information processing. These handlers are pieces of code that are activated during message reception by a component of the system. They can activate various response mechanisms, or initiate calculations that are designed to respond to specific events. These handlers feature underlying mechanisms that are closely involved with unpacking and processing messages or events. The handlers are programmed with predisposed response tactics. Different handlers deal with different types of events (Stallings, 2017).
* Asynchronous Message Passing – *Benefits*
* iSpace aims to traverse dangerous environments. This requires iSpace’s system architecture (LRTOS) to be able to quickly respond to various incoming messages. The following information will discuss the value of having asynchronous message passing in a system that relies heavily on real-time endeavors. The asynchronous characteristic of this message passing mechanism allows processes to carry out their necessary functions at the same time. As the aerospace environment can exponentially present extraneous variables to the system, it is crucial that each unit can effectively coordinate information. AMP brings with it several benefits to the system architecture. AMP is compelling to use because of its ability to have multiple messages relayed at the same time. This type of message passing mechanism can be useful in scenarios that require quick transmission of multiple pieces of information. Examples of these pieces of information are sensor readings or status updates.
* Fault tolerance is the ability of iSpace’s Linux-based Real-Time Operating System to continue its normal operation in the face of hardware or software failures. A high measure of fault tolerance necessitates levels of redundancy. Such redundancy tactics can include spatial or information redundancy. Spatial redundancy involves the configuration of physical components to do the same job, or designing one component to be the backup in case the primary component fails. Information redundancy copies the data in question such that errors can be observed and fixed. Fault tolerance is intended to increase the reliability of a system (Stallings, 2017).
* Asynchronous message passing can encourage fault tolerance through the decoupling of components. This implies that each subsystem can continue to operate independently even if one subsystem fails to transmit or process the correct information. By decoupling the sender and receiver, the system has greater flexibility in the use of messages (Stallings, 2017). Decoupling can allow effective asynchronous communication. Because of this, the iSpace operating system architecture can ensure that information is properly transmitted between subsystems. In the grueling environment of aerospace, one cannot underestimate the value of speedy message passing.
* In the context of a spacecraft avoiding space debris, asynchronous message passing allows the specialized subsystems to function atomically. By doing this, each operation is executed in an isolated fashion and no other processes can intrude. This avoids race conditions and competition for resources. For example, debris sensors and propulsion components can carry out their functions independently of one another. Neither operation must wait for a signal from the other (Stallings, 2017). This non-reliance between components allows for swift hazard detection and reaction.
* AMP also facilitates urgent real-time responsiveness and scalability. This is mandatory in a rapidly changing environment such as aerospace. The Linux-based embedded RTOS (LRTOS) has qualities that allow it to react to these rapid changes. By employing asynchronous message passing, the LRTOS picks up information such as sensor readings and exchanges messages in a proficient manner. While one component processes maneuvering trajectory, another could be analyzing the current state of the system. These devices in conjunction with AMP make it possible to analyze and interpret multitudes of information bits.
* AMP’s isolative nature makes it feasible to implement more levels of complexity into its existing components and add new components that will enhance the system. When a new feature or component must be installed, engineers are not forced to modify the original system from scratch. Instead, a new feature can be implemented into the system because of the decoupling mechanisms involved in asynchronous message passing. Processes do not have to interweave and wait for routes and destinations. Each process can operate independently, therefore the new components that are added to the system are not reliant on their predecessors (Stallings, 2017).
* AMP also provides flexibility in terms of its message processing techniques. Handlers can be configured to assign priorities to the reception and queuing of messages. iSpace implements immediate processing as its primary mode of asynchronous message passing in its spacecraft. This mode is simple and ensures speed is its primary focus. The expeditive nature of this mechanism introduces predictability of the surrounding environment. If messages are transmitted immediately one after the other, it can allow for an accurate portrayal of the data from the environment surrounding the spacecraft (Fanfakh, 2016).
* Asynchronous Message Passing – *Challenges*
* While AMP brings several benefits to the table, it also brings challenges. One of these challenges is adapting to the time constraints provided by the environment that is to be traversed while minimizing overhead issues. Overhead is the additional resources taken up by implementing a mechanism such as asynchronous message passing. Overhead can greatly reduce the responsiveness of a system. The AMP mechanism involves the packaging and unpackaging of information into understandable forms by different sending and receiving parts of the system (Stallings, 2017). By adding these extra steps, a mission can be jeopardized if engineers of iSpace are not careful during the mechanism design process.
* Entities such as spacecraft and satellites are often predisposed with an estimation of necessary resources predicted much earlier than the launch date. While it is true that this estimate is constantly changing, engineers must consider the appropriate measures to facilitate a successful endeavor. A mechanism such as AMP is another added variable that these engineers must account for when managing resources. Such resources can include power and memory. Asynchronous message passing requires the reading and writing of memory during message passing, which can prove to be too exhausting to resources. The Linux kernel is constantly switching between processes, and the resources required to add an element of AMP may not be efficient especially in such a time-sensitive environment (Stallings, 2017).
* While AMP can enhance system communication by independent process operation, it can degrade communication as well if the design of the AMP mechanism is prone to bottlenecks. When multiple processors must access the same memory at the same time, a bottleneck occurs. There are a few instances where this is applicable such as if the navigation system and the propulsion components are both competing for the same resource concurrently. Enough memory must be configured within the system to achieve effective asynchronous message passing otherwise the LRTOS is unable to carry out functions in a timely manner due to insufficient memory space. In the case of aerospace, large volumes of data are transmitted over huge distances. AMP must be implemented in a way that ensures reliable data transfer without congestion. Congestion occurs when the arrival rate is larger than the departure rate (Stallings, 2017). It can be challenging for engineers at iSpace to design an AMP architecture that aligns with mission objectives while maintaining the LRTOS integrity.
* Asynchronous Message Passing – *How it Supports the Environment*
* The unpredictability of outer space lends itself to the possibility of a catastrophe. To minimize this probability, iSpace’s LRTOS aims to properly compartmentalize the information it is confronted with. Asynchronous message passing gives way to snappy collection and distribution of data. For instance, a satellite’s LRTOS can be configured with AMP that implements priority-based messaging. In priority-based messaging, tags are attached to each message that is sent and received. The message tag can be used to portray the message type. The tags are used to ensure that the corresponding components receive and process the correct information in the right order. The message tag consists of bits which indicate the priority of a message (Karlsson, 2005). This mechanism is used to ensure that higher priority messages are sent in front of lower priority messages. In a satellite embedded with the LRTOS, this mechanism provides efficient methods of storing and processing incoming data.
* In the architecture of this system, fairness and minimizing average response time are not the goal. Instead, iSpace aims to ensure that hard real-time tasks complete or begin by their deadline and that as many as possible soft real-time tasks also complete or begin by their deadline. An example of a hard real-time task is the launch window of a spacecraft while an example of a soft real-time task is the transmission of data collected by a rover. The characterizations of what deadlines are considered hard or soft deadlines change with the mission objective. Real-time applications require response times in the range of milliseconds (Stallings, 2017). In spacecraft, real-time data processing is crucial in maintaining system health and manipulating system actions. Asynchronous message passing can support real-time data processing by simplifying the messages relayed between different parts of the system. For example, AMP can drive the transmission of sensor readings and command signals between separate components of the LRTOS in real-time. This allows for prompt decision-making by the LRTOS and response to the constantly changing mission environment.
* Due to the volatile nature of aerospace, the Linux RTOS is constantly in need of modifications. AMP ensures that the separate components can decode sent messages even if they are sent by an unrelated unit. The interweaving of specialized components becomes easier due to the independent exchange of information within the system. In other words, because each piece of hardware is not dependent on the architecture of another, new hardware can be implemented with minimal complexity (Stallings, 2017). This allows for real-time updates and advancements in protocols. As a mission progresses, it may require these procedural updates to keep up with the requirements of the mission. This simple hardware integration makes it feasible to carry out difficult tasks in a cost-effective manner.
* Asynchronous Message Passing – *How it Supports Process Communication*
* The rovers, satellites, and spacecraft of iSpace must have built within them an effective method of communication between processes to minimize risks. AMP is a suitable concurrency mechanism that allows for systems to carry out their tasks without waiting for an immediate response signal from their counterparts. With the goal of maximizing the overall throughput of the LRTOS, asynchronous message passing implements effective process communication behaviors. This also leads to the overall reduction of idle time by the system. The scarcity of resources in outer space leaves little room for error. The asynchronous sending and receiving of messages based on priority allows for the proper allocation of resources. This leads to each of the compartments within the system having enough access to those resources and in turn, being able to have enough time with those resources because they do not spend too much time on encoding or decoding messages. AMP allows transmission of messages in a way that multiple components can understand (Fanfakh, 2016).
* Asynchronous communication makes it possible to overlap computation with communication. Synchronous communication, on the other hand, can potentially reduce the latency of communication when the system or application is otherwise idle. Decoupled communication, which requires no direct association between the producers of information and its consumers is a characteristic of asynchronous message passing (Murata, 2003). Decoupled communication allows each component to operate atomically. This enhances process communication by allowing different parts of the system to communicate when necessary. This means that rather than communicating something such as reception confirmation to a sender, a component can take that time to communicate with another desired component. This leads to an effective method of communication and resource conservation.
* Asynchronous message passing serves as a strong mechanism for communication between processes within the Linux RTOS embedded into spacecraft. Processes running on the LRTOS communicate by exchanging messages asynchronously. This allows them to share data and coordinate activities effectively. The asynchronous message passing mechanism enables processes to communicate efficiently and reliably, even in the face of significant communication delays or disruptions.
* Sockets are process communication tools that can be incorporated with asynchronous message passing within the LRTOS system. Sockets enable communication between a client and server process. A socket can be considered an endpoint in a communication link. A client socket in one computer component uses an address to call a server socket on another computer component. Once the appropriate sockets are engaged, the two units can exchange data. In the case of AMP, these sockets must have a non-blocking function which allows the system to issue a read or write operation and continue with other processes without waiting for completion (Stallings, 2017). This is an integral part of the concurrency mechanism in this embedded aerospace environment.
* Asynchronous Message Passing – *How it Supports Process Synchronization*
* Despite the misleading name, AMP can support process synchronization through a variety of ways. AMP’s non-blocking nature and its message queueing structure allow processes to synchronize their actions by letting them continue execution while waiting for messages or performing message-related operations. This mechanism is useful in the case of iSpace’s embedded LRTOS because of its allowance of processes to continue firing at their own rate, thereby allowing a synchronized environment. In the scenario of an iSpace rover with an objective to collect samples of land, the rover’s LRTOS must be able to navigate between the different functions of each component of the system. This could include navigation of the rover’s wheels synchronized with its arm movements. If it weren’t for AMP’s non-blocking synchronization, the rover may spend far more time completing the mission.
* Semaphores are synchronization tools used in concurrency mechanisms to allocate resources. Processes can send messages to request access to semaphores asynchronously, indicating their need for a shared resource. iSpace’s mechanism allows multiple processes to do this at the same time. Polling in semaphores implemented through AMP aims to optimize responsiveness. By checking for semaphore availability without blocking, processes can access shared resources while maintaining responsiveness and parallelism in a concurrent environment. Processes can hold or give up the semaphore to indicate if they must utilize a resource. Semaphores in conjunction with asynchronous message passing prevent race conditions and enforce mutual exclusion (Stallings, 2017). In the context of a rover collecting samples, semaphores and AMP can synchronize resource access between different components of the rover's system. For example, a component of the rover's system controlling navigation could send a message involving a semaphore request to access the rover's wheels while traversing terrain.
* Mutexes are synchronization tools that control access to shared resources by allowing only one process or thread to access the resource at a time. When implemented with asynchronous message passing, mutexes help to make sure processes synchronize their access to shared resources efficiently (Stallings, 2017). Processes can send messages asynchronously to request a mutex on a shared resource. This allows processes to continue execution without waiting for a response, enhancing concurrency and synchronization within the LRTOS. When a process requests a mutex, it waits until the mutex becomes available. Asynchronous message passing ensures that processes can check the availability of the lock without blocking, allowing them to perform other tasks while waiting (Stallings, 2017). This can greatly reduce the time needed to carry out a mission objective such as in the case of a rover collecting samples.
* Monitors – *How they Work and How they are Used*
* A monitor is a programmed code that has a similar function to semaphores. Monitors are easy to control in the LRTOS, and they have been used in several programming languages such as Java. The monitor has also been used as a program library which allows developers to add variability to the locking system. This can be seen in a linked list where a developer can put a monitor lock on any element in a list or they can lock all linked lists with just one lock. A monitor has a set of procedures, an initialization sequence, and data. The data can only be accessed by the monitor’s procedures, and a process can enter the monitor by initiating one of those procedures. Only one process can execute in the monitor at a time while other processes attempting initiation are blocked. In turn, shared data is protected inside of a monitor (Stallings, 2017).
* Monitors direct process threads to wait before executing. Monitors utilize wait and notify procedures, where threads can hold until a certain condition is signaled by another thread. Condition variables are within the monitor accessible only by itself. Condition variables are a type of data in monitors. The following is an example:

cwait ( c )

csignal ( c )

The first function is the halt of a process calling execution, and the monitor can be accessed by other processes. The second function resumes the execution of a process that has been halted (Stallings, 2017).

* A monitor can be thought as the doorman, and processes must check-in with the monitor before they can continue their actions. The processes have only one entry point, ensuring resource protection. If a process is currently in use of the monitor, other processes are queued. If a process attempts to change the memory area of the monitor, the hardware considers this a malfunction and control is given back to the monitor. The function is then aborted, the LRTOS is notified, and processing can continue. Some machine instructions are designed to have privileged access by the monitor only. If a processor sees one of these instructions while executing a function, it will assume malfunction and the monitor is able to take over (Stallings, 2017).
* Monitors can be seen used in cases such as the dining philosophers problem. In this problem, a monitor manages access to five forks for five philosophers. The monitor uses condition variables, indicating availability of the forks. When trying to pick up both forks, a philosopher will initiate the monitor’s procedure to pick them up, and if either is unavailable the philosopher waits. This lets other philosophers utilize the monitor. When the philosopher is done with the forks, another procedure is initiated indicating availability of the forks. This use of monitors prevents deadlock. Deadlock is an instance where the system is at a halt and no progress can be made. Monitors ensure that forks are picked up individually and both are usable before the philosopher starts eating (Stallings, 2017).
* In a sense, monitors are comparable to traffic lights and are used to avoid race conditions. They direct which processes can access its resources at what time. This avoids any process conflicts and prevents race conditions. A race condition occurs when multiple processes or threads modify the same data, and the result depends on the order of their actions. For example, if two processes try to change the same variable, the last change is the one that stays. In this case, the process that updates the variable second gives the result (Stallings, 2017). Usually this will result in unwanted outcomes, so monitors are used to avoid situations that cause errors such as incorrect data manipulation within the system’s memory.
* Monitors can implement atomicity as one of its features. This operation can be used to allocate and conserve resources efficiently while minimizing interface time. An atomic operation is executed without interruption and without interference. In LRTOS monitors, the variable being operated on is locked from other threads until the process is completed (Stallings, 2017). This mutual exclusion is crucial in cases where data must be conserved such as in sample collection. Monitors in conjunction with atomicity are used to enable efficient execution of important parts of code. This type of all or nothing principle is also used to make sure no time is being wasted by unnecessary interruptions.
* Not only are monitors used to protect the initial granting of access to a data source, but they also regulate how long a process has with a resource. Monitors can be configured with timers. A timer is used to prevent a single process from taking up a lot of time with a resource such as data. The timer is set at the initial stage of process execution. If the timer expires, the process is stopped, and control returns to the monitor (Stallings, 2017).
* Monitors – *Benefits*
* An advantage that monitors have over something such as semaphores is that all synchronization functions are left to the monitor. Because of this, it is easier to confirm that synchronization has taken place correctly. When a monitor is correctly programmed, access to the shared resource is correct for access from all processes (Stallings, 2017). It is of utmost importance that data collected by iSpace is protected by numerous mechanisms. Monitors enhance iSpace’s ability to ensure this protection by minimizing data corruption through means of mutual exclusion. In a layered enterprise where multiple tasks and processes run concurrently while adhering to stringent time constraints, monitors maintain integrity of sensitive information such as flight logs or sensor data. By using monitors, iSpace’s critical mission activities ensure adequate responses to unforeseen external events. Monitors allow iSpace’s units to control the timing and sequencing of tasks, optimize overall system capability, and react to adverse environmental conditions encountered during aerospace exploration.
* The flexibility of monitors makes them a suitable concurrency mechanism in iSpace. As mission objectives evolve, monitors provide a manageable approach to controlling concurrency. Developers can implement new sections of code into the monitors as needed, allowing iSpace to easily modify its designs to keep up with mission goals. Constant modification is an inevitable necessity to face real-time tasks especially in outer space. Monitors make it easier for iSpace engineers to find flaws within the system. This is because the integrity of important data within monitors is conserved through hardware-monitor interaction. Proper monitor implementation and its controlled access to shared resources allows maintenance of iSpace to be less time consuming. Teams can isolate these issues quickly, and debug code more efficiently because the flaws are easier to find (Stallings, 2017).
* Tagging these bugs increases reliability of iSpace’s functional units. The time it would take a rover to identify the problem with a rover’s movement may take substantially longer if that information was not protected by a monitor. Because monitor code can be modified and its interface between hardware allows data protection, iSpace’s hardware can be reused for different purposes. This effectively reduces costs because engineers are not forced to change every piece of hardware (Stallings, 2017). Overall, monitors as concurrency mechanisms in iSpace's embedded systems provide benefits such as real-time response, adaptation, and flexibility. These advantages contribute to the efficiency and security of iSpace's operations.
* Monitors – *Challenges*
* It is possible to make mistakes in the synchronization function of monitors. For example, if either of the signal functions in the buffer monitor are deleted, then processes entering the corresponding condition queue are unable to carry out function (Stallings, 2017). iSpace engineers must be careful to differentiate between what is considered important shared data and what is not. Corrupted data can certainly jeopardize the mission, and it may take developers ample time to continue modifying the LRTOS monitors depending on the mission tasks. While monitors are effective, another challenge in implementing them involves the proper timer setting. Processes cannot be allowed to monopolize use of the monitor, but they need enough time for atomic execution. Finding the correct setting for the timers may be difficult to find for engineers due to the intense and unpredictable nature of aerospace.
* When it comes to implementing monitors within iSpace systems, overhead may be encountered due to energy and time constraints. A scenario could arise where there are not enough resources such as CPU power to carry out full function. This may include the overhead caused by using condition variables or the timer. Since priority is given to maintaining integrity of units such as spacecraft, secondary resource allocation directed towards monitor function may not be feasible. Energy must be conserved as much as possible in space systems. Monitors use up energy and take up some CPU time (Stallings, 2017). If the unit comes across some unexpected circumstance, the power used by monitors may force the rest of the mission duration to be cut down.
* Using monitors can be devastating to the system in cases of emergency. If a monitor fails, all the information it contains may be corrupted. Processes would no longer be mutually excluded, and the system would not be able to carry out its function. If a monitor fails to do its job, or if a timer malfunctions within the monitor, it could result in catastrophe. Several problems can occur depending on which part of the monitor’s code is flawed (Stallings, 2017). An entire spacecraft can lose all redundancy and reliability because the inefficiency of one monitor leads to the inability of processes between different components to communicate. As mentioned, processes must be able to communicate effectively in such a strenuous environment.
* Monitors – *How they Support the Environment*
* As discussed, the LRTOS architecture is specifically designed with regards to optimizing the function of spacecraft, satellites, and rovers in their appropriate settings. Monitors as concurrency mechanisms are integral for supporting the environment of the Linux-based Real-time Operating Systems (LRTOS) embedded in units within the iSpace enterprise. Monitors allow the company to efficiently manage shared resources within its spacecraft components. These resources include hardware such as propulsion systems and scientific instruments. Coordination of access to these resources prevents timing conflicts. A monitor’s ability to give data access to appropriate processes can be the deciding factor in whether a unit such as a spacecraft will be preserved in outer space. The timer built into the monitor is an important supporting feature because it can make sure tasks are executed in a deadline-oriented manner. Navigation processes are allowed to fire off one after another because of this and the system can make quick reactive decisions (Stallings, 2017).
* If an error does occur, it can be easily detected by the LRTOS due to this monitor mechanism. The tagged regions of information, if corrupted, allow for the system to point out the mistake and recover. If a failure occurs, the presence of a monitor can help to contain the amount of leakage there is to the whole system. For example, if a rover were to encounter a hardware failure with one of its sensors, having a monitor could reduce the impact on the rest of the system. The monitor can observe where the error has occurred and make sure other processes do not attempt to interact with the flawed region (Stallings, 2017).
* Monitors can also help manage power consumption and distribution within rover systems by directing interactions with power components such as solar panels or batteries. By facilitating the activation and deactivation of these devices within monitored regions, monitors ensure efficient utilization of available power resources. In turn this will accommodate for any mission deterring events that have made that power scarce. Increasing mission duration is imperative to solidify the chances of success. Time constraints are already severe as it is, and monitors allow for proper distribution of access to data and various sources.
* Monitors allow cooperation between different processors within iSpace systems. This gives way to effective information exchange and prompt decisions. They contribute to resource allocation, fault tolerance, and survivability.
* Monitors – *How they Support Process Communication*
* Communication between components is essential for managing simultaneous access to shared resources and ensuring data resiliency within spacecraft systems. Monitors have built-in features such as mutexes and condition variables which direct access to shared data structures (Stallings, 2017). This prevents data corruption and resource contention which is crucial in maintaining the integrity and reliability of spacecraft operations. iSpace’s system monitors that act as concurrency control mechanisms implement pipes to give more layers of inter-process communication. Pipes can be used to communicate data between processes. A pipe is a buffer between two processes to communicate with one another, while keeping in mind the producer–consumer problem. The capacity of a pipe to hold information in bytes is determined up creation of the pipe. If there is room for a process to write into a pipe, the write is executed right away or else it is blocked (Stallings, 2017). iSpace’s LRTOS enforces mutual exclusion so that only one process can access a pipe at a time. iSpace engineers have implemented named pipes instead of unnamed pipes because only related processes can share unnamed pipes. In named pipes, processes can be unrelated, leading to cohesion between components of the system (Stallings, 2017).
* Monitors ensure that processes do not interfere with other processes. As they are carrying out tasks, processes wait for granted permission to required resources. This mechanism effectively allows communication between processes and enables processes to execute atomically if they abide to time constraints (Stallings, 2017). In the case of a rover collecting samples, a monitor’s ability to implement timers provides indirect communication between two processes competing for the same resource. One analyzing process needing access to the rover’s collected sample data must wait until another process has fully secured that data.
* Monitors ensure that processes accessing shared resources do so in a communicated manner, preventing concurrent access that could lead to data corruption or inconsistency. By covering critical sections of code within monitored parts, processes can request access to shared resources and wait until they are granted permission by the monitor before proceeding (Stallings, 2017). This synchronization mechanism effectively coordinates communication between processes, ensuring that data is accessed and manipulated in a controlled and orderly fashion.
* In iSpace’s units, multiple processes can run concurrently despite their need to access a shared sensor data buffer or perform data analysis tasks. Each process requests access to the buffer through the monitor, and the monitor grants permission to one process at a time. Monitors utilize notification mechanisms, such as their condition variables, to signal other processes when a shared resource becomes available or when certain conditions have been updated to relevant levels. This enables processes to coordinate their actions based on the current state of the shared resources while constantly being able to check for condition requirements (Stallings, 2017). Facilitated communication and coordination among concurrent tasks within a spacecraft is a useful product of the LRTOS monitors. They provide stability by reducing confusion and chaos between process tasks of reading and writing. Overall, monitors in an LRTOS support communication by enforcing single-file access to important data and allowing inter-process notifications.
* Monitors – *How they Support Process Synchronization*
* To be useful for concurrent processing, the monitor must include synchronization tools. These tools include mutexes and condition variables. Beneficiating from mutual exclusion is important for the LRTOS. Mutexes can be considered locks that processes need to access a shared database. Processes must get the lock upon entry and get rid of it upon exit, and they can only do this one at a time. When implemented with monitors, mutexes provide a way for orderly interactions. When a process is done with a resource, the next process can fire off immediately after the mutex is released. When a process has the mutex, which can be considered a key, it is safe to resume its operations within the monitor. This allows for consistency between processes that interact with the shared data (Stallings, 2017).
* Monitors also support process synchronization through fault tolerance by means of recognizing exactly where a fault has occurred. Because a process must acquire a mutex, any corruptions to the data can be seen immediately after the mutex is released and the monitor has control of the unoccupied resource. Maintaining isolation between concurrent processes and their interactions with shared resources is an important synchronization feature provided by monitors. When a fault occurs, the mutex no longer lets other processes access it. This allows for the conservation of resources giving rise to further synchronization (Stallings, 2017).
* Condition variables can be considered traffic signals within monitored areas for processes. They indicate to a process when it is time for the process to execute. Until the condition is met, the process continues to wait. This reduces congestion by decreasing the rate of traffic (Stallings, 2017). This can be seen in rover exploration one component of the LRTOS runs sample analyzation, while another can determine new samples in comparison to previous samples. In this case, the monitor enforces only one process being able to execute on data related to the sample at a time by utilizing these condition variables. When new samples are added to the system, it starts a chain which leads to the initialization of the recording process. These techniques in conjunction with the others discussed, give iSpace’s LRTOS the ability to persevere in the aerospace environment safely.

## OS Security Risks and Mitigation Strategy

* Operating System Evaluation
* Several areas of an operating system designed for aerospace exploration must be evaluated prior to launch. The following areas are critical:
  + Resource Management
  + Real-Time Response
  + Data Preservation
  + Hardware Resilience
  + Network Security
* *Resource management:* The host OS must ensure proper resource (ex: power, memory) allocation amid traveling outer space. A proficient OS can distribute resources such that the priorities of the objective are encouraged.
* *Real-Time Response:* An OS in units such as spacecraft must be able to quickly and correctly respond to external variables.
* *Data Preservation:* An OS in outer space must be able to store and protect collected data effectively.
* *Hardware Resilience:* The extreme conditions of the aerospace environment call for durable hardware components.
* *Network Security:* Integrity of the OS depends on network security levels. This OS must not be susceptible to any type of network failure.
* Risk Assessment
* A risk is described as a threat severe enough to jeopardize the totality of a mission, and therefore must be dealt with. Risks are uncertainties that could impact mission success. iSpace’s LRTOS design has been dissected to observe areas of vulnerability. After risk assessment, the enterprise has implemented protocols and regulations to ensure safety of iSpace functional units. The aforementioned areas of the operating system have been deemed most suitable for checking and mitigation. These areas have been determined as most suitable for assessments because the enterprise prioritizes the integrity of its spacecraft, satellites, and rovers. Each of these areas, if given the proper attention, can provide a strong foundation for any mission. The following is a discussion of assessed risks and their mitigation strategies for iSpace’s LRTOS.
* *Resource Monopolization Risk*
* The OS is responsible for controlling the use of a computer’s resources, such as I/O, main and secondary memory, and processor execution time. Processes’ interaction can be classified by the way they interact with each other. Processes can either be unaware of each other, they can be indirectly aware of each other, or they can be directly aware of each other using primitives. In processes that are unaware of each other there is competition, but the process outcomes are independent of one another. Even though these processes are independent and do not work together, the operating system must still consider the competition for resources. This can be seen if different programs attempt to use the same disk or file. In processes with indirect interaction there is cooperation by sharing, and the outcome of one process may be dependent on received information. In direct awareness there is cooperation by communication, and the outcome may also depend on received information. These processes are designed to work cohesively, and they communicate by process IDs (Stallings, 2017).
* Each of these cases present potential issues of deadlock and starvation of resources within the LRTOS. When multiple tasks are running at the same time, they can interfere with each other if they’re trying to use the same resource, such as memory. If multiple processes must access the same resource and they are unaware of the others’ existence, each process should not affect the original resource integrity. Examples of these resources include printers or the system clock. While these tasks don’t have direct communication, one task can still affect another if one is forced to wait for a resource. In some cases, it is possible that a task may be blocked indefinitely (Stallings, 2017).
* Deadlock can be seen when the operating system allocates two resources (R1 and R2) to two tasks (P1 and P2). If each task needs both resources to complete its job, each task is waiting for the other resource that it doesn't have. Neither task will give up the resource it holds until it gets the other one it needs to finish its job. Because each is waiting for the other to release a resource, neither can move forward. This situation is called a deadlock, where both processes are stuck (Stallings, 2017).
* Starvation can be seen if 3 tasks (P1, P2 & P3) need to use the same resource (R). If P1 is using that resource, P2 and P3 are forced to wait. When P1 is finished, the OS will allow P2 a turn. If P1 requires that same resource again, the system may not allow P3 to ever have a turn. It is different from deadlock because while the resource is being used by multiple processes, one or more processes still do not have access to it (Stallings, 2017).
* iSpace considers the risk of certain tasks occupying too much CPU time to be unacceptable. This can be seen in cases involving severe time constraints. For example, a spacecraft taking evasive action may cause the system to allocate all its resources to one area. This leaves the rest of the components to be deprived of important resources such as power. Other components affected may be communication related. One sequence of evasive maneuvers could be detrimental to the rest of the operation if proper mechanisms are not in place to account for these unforeseen situations. A scenario could arise where a rover has allocated too much of its processing power to analyzing a sample, leaving its navigation capabilities at bay. A satellite, if forced to recognize an unforeseen circumstance, may not have enough power to store future information.
* Prolonged resource monopolization can start a cascade of component failures. Non-urgent tasks such as data preservation or routine maintenance checks can be significantly delayed, leading to inability to complete the mission at hand. Often, the most important objective of iSpace is storing and collecting new data. This objective can be disrupted in a multitude of ways in outer space due to its unpredictability. In the case of a rover collecting samples, resources are allocated such that the rover has sufficient power to maintain full duration of an objective. If an unforeseen circumstance such as a storm is to develop, the rover might be forced to reallocate resources if it becomes too difficult to navigate the environment. If the rover puts too much power into moving its wheels, it may not be able to keep up with analyzing samples during the evolving conditions of the storm because of its overall reduced throughput.
* *Risk of Data Loss or Data Corruption*
* Buffer overflow occurs when a program attempts to write more data to a temporary storage area than it can hold. This may cause the new data to overwrite previously written data by overflowing neighboring buffers. These locations may contain program variables or program control flow data such as return addresses and pointers to previous stack frames. This can cause data corruption and crashes. The unwanted overwriting can delete valuable memory that is necessary information for the program to know what its next move is. If this process is unchecked, the system will continue to unknowingly and unwillingly corrupt its own collection of data (Stallings, 2017).
* A bug in a single routine, such as an interrupt handler, could damage process control blocks, which could destroy the system’s ability to manage the affected processes. Even the smallest errors in coding could cause the system’s ability to store information to fail. For example, a bug in the file management component of the operating system can cause files to be unwillingly deleted or corrupted while moving or copying those files. Memory Lifecycle (MLC) bugs occur if memory is not allocated, dereferenced, or freed adequately. If memory is not freed when it should be it can lead to data corruption of critical data stored in memory (Stallings, 2017).
* High latency in communication can contribute to data corruption or data loss. Interrupt latency is the time a system takes to respond to an interruption and begin executing an interrupt service routine. Task switching latency is the time an OS takes from when a thread is ready to when it is executed. An OS attempts to minimize memory footprints by enabling components to configure memory only when necessary. Latency in an operating system can cause disfunction especially if it is meeting stringent time requirements. If a system takes too long to process a certain amount of data, it may not have ample time to store the rest of data that needs to be gathered. High latency situations may cause the kernel to display unpredictable behavior (Stallings, 2017).
* It is imperative for iSpace to address these issues of possible data corruption because of its mission goals. Data being transmitted over vast distances opens the possibility of data loss. This can be seen in the example of a rover collecting samples. If communication latencies arise, the rover may not be able to collect enough samples before environmental hazards come about. Real-time communications between a rover and its base station must be as strong as possible. If movement commands exceed the buffer size limit in a rover, it may cause the rover to move sporadically and endanger collected sample data.
* Because data preservation is of utmost importance for iSpace, it is important for iSpace engineers to consider the impact of software bugs. A bug in navigation software could lead to an impact on the rest of the system. For example, a rover could fail to traverse terrain due to a glitch in the system determining its orientation. This glitch would then cause the rover to navigate its surroundings incorrectly. A cascade of power loss and incorrect sample collection would occur because of one point failure. If the bug is severe enough that the rover is no longer responsive, it could be an end to the mission itself.
* *Risk of Hardware Failure*

- Hardware failure is often a risk that is accounted for in many operating systems. It is generated by some sort of imbalance such as power failure or memory parity error. Memory errors and device malfunctions can occur while a computer system is running. The availability of an OS is the time the system is available to execute users’ requests. Availability is the likelihood that an operating system is working properly under a set of conditions at a certain point in time. The time during which the system is not available is called downtime while the time during which the system is available is called uptime. A fault is a state of error that stems from component failure, design error, or interference from the environment. A fault can be a defect in a hardware device such as a short circuit or broken wire. Faults can be permanent or temporary. A permanent fault always exists after its occurrence. A permanent fault requires maintenance and will persist until it is repaired. Examples of this can include burnt communications components (Stallings, 2017).

- A temporary fault is not always present for each operating condition. A transient fault is a fault that occurs only once. It is a single glitch that happens because of interferences such as power surges. Intermittent faults are random. These faults occur at multiple and unpredictable times. This can be conceivable in the case of a loose connection causing an issue every so often (Stallings, 2017). Over time, a piece of hardware can malfunction due to normal wear and tear. It can also malfunction due to overuse. If a single component bears the burden of facilitating most of the load, it may cease to work properly after an incident of extreme exertion. If hardware is not properly manufactured it can be prone to physical damage and errors. Any damage may reduce the functional duration of hardware units.

- Prolonged hardware failure can lead to irreplaceable data loss. If a piece of hardware that contains valuable information gets damaged, that data may be inaccessible. Prolonged system downtime can also lead to a loss in productivity and profit. Hardware failure can result in surmountable financial repercussions. It may not be feasible for a company to remanufacture components if they are costly. Hardware failure can also result in the discontinuation of operations if the failure causes severe enough damage. Not only would industry competitors see this as a weakness, but the company would lose the trust of its own community. Hardware failure may be the most catastrophic of events depending on the circumstance.

- Hardware failure of the LRTOS can certainly be one of these catastrophic events. Not only could a single-point failure jeopardize the mission task at hand, but it could also make it impossible to continue with further missions. A hardware failure in the case of a spacecraft could include a power system failure. A malfunctioning battery failure could lead to a loss of power and render the spacecraft unusable. Without power, critical systems like navigation and communication would stop functioning. A failure in the communication system, such as a malfunctioning antenna, could cause an outcome where the rover is lost permanently.

- A glaring possibility of hardware failure is a spacecraft’s inability to launch. If severe enough, the damage could be disastrous. The navigation systems of the functional units of iSpace are crucial to any mission endeavor. The success of an operation depends on their reliability. Even a perceivably small error, such as a single sensor failing to give accurate temperature readings, could change the course of a mission. Another example can be seen in the case of a malfunctioning camera. If the camera is not able to process images, the efforts become futile. If iSpace wishes to prioritize the safety of its astronauts, it is crucial to implement proper life-support systems such as oxygen sensors.

* *Risk of Network Failure or Data Breach*
* Network failures and data breaches are common concerns amongst many major corporations. Computers are linked to networks, and information is moved from one computer to another using the network. The transfer of data from one application to another involves two steps. First, the data must reach the appropriate computer, and then the data must find the appropriate program on that computer. The physical layer deals with the actual physical connection between a device sending data and its connected network. This layer deals with describing how the transmission medium is used with what signals, and the speed of data transfer. The network access layer manages the sharing of data between an end device, such as a server, and its connected network. The sending computer must provide the network with the correct address of the destination computer so the network may handle the data accordingly (Stallings, 2017).
* When two devices are attached to different networks, procedures are needed to enable data to cross multiple interconnected networks. The Internet Protocol (IP) is used at this Internet layer to provide the routing function across multiple networks. This protocol is implemented in routers as well. A router is a processor that acts as a bridge. Its main function is to move data from one network to the other on a route from the source to the destination end system. Network device drives are used to supervise network interface cards and communication ports linked to network devices such as routers. Network namespaces create a division of system resources involved with networking. This allows each network namespace to have its own network devices and IP addresses (Stallings, 2017).
* Buffer overflow can be exploited by attackers to crash a system or insert custom code that allows them to gain control of the system. There are three types of intruders: masqueraders, misfeasors, and clandestine users. Masqueraders are unauthorized individuals who infiltrate a system’s access controls to use an authorized user’s account. Misfeasors are authorized, but they misuse this authorization by accessing untouchable data. Clandestine users are individuals who gain control of a system to evade audits. Some programs that exploit buffer overflow can be referred to as malicious software. This code can hold malicious instructions and can be used during buffer overflow when memory is leaking. When done as a purposeful attack, the control transfer could be to any code of the attacker’s choosing. This can result in the ability to execute any code with the authority of the attacked process. Buffer overflow attacks are common and dangerous (Stallings, 2017).
* iSpace relies heavily on its LRTOS and networks to support its operations. Network failures and data breaches can harshly impact enterprise functions and dissipate important information. iSpace’s confidentiality of aircraft design and mission planning is kept at high regard. If iSpace systems were to be intruded upon, competing companies could steal intellectual property. This could involve the theft of aircraft blueprints or mission objectives. The enterprise obliges several contractor deadlines. It cannot afford to experience the delays caused by malicious intrusion.
* Buffer overflow vulnerabilities can cause the complex interconnection of iSpace’s hardware and software to be corrupted and crash. Masqueraders, misfeasors, and clandestine users can take advantage of buffer overflow. They can falsify authentications and corrupt precious operations to defame iSpace or take the company’s blueprints. Misfeasors could insert malicious code into components such as propulsion systems, leading to unpredictable behavior. This would ultimately risk safety and may result in catastrophe. An unauthorized user could also nullify mission success by interrupting communication channels.
* *Risk of Improper Task Scheduling & Prioritization*
* Any processor can perform scheduling, which complicates the task of enforcing a scheduling policy. Multiple processors scheduling simultaneously also makes it more difficult to assure that corruption of the scheduler data structures is avoided. In kernel-level multithreading, a system can schedule multiple threads from the same process at the same time on multiple processors. The scheduler is one of the most important features of a real-time system. Real-time computing is defined as a type of computing where correctness of the system depends on the timing of produced results, not just the logical results itself. Most tasks in real-time systems have degrees of urgency to them. These tasks are attempting to control or respond to events that take place externally. Because these events occur in the real world, a real-time task must be able to keep up with the events with which it is concerned. When a deadline specifies either a start or completion time, such a task can be considered hard or soft. A hard real-time task must meet its deadline. A soft real-time task has a wanted deadline, but it is not mandatory. Even if a soft real-time task has missed its deadline, it should still be scheduled. Another characteristic of real-time tasks is whether they are periodic or aperiodic. An aperiodic task has a deadline by which it must finish or start (Stallings, 2017).
* How deterministic an operating system is dependent on its ability to perform operations at fixed, predetermined times or within a certain amount of given time. When multiple processes are competing for resources and processor time, no system will be fully deterministic. In a real-time operating system, process requests for service depend on external variables. A deterministic operating system must execute requests and respond to interruptions promptly. The time needed to handle the interrupt and begin execution of the interrupt service routine (ISR) impacts determinism as well. If execution of the ISR requires a process switch, then the delay will be longer than if the ISR can be executed within the context of the current process (Stallings, 2017).
* An improper scheduling mechanism can result in the inability of a system to correctly and quickly process information. The reliability of LRTOS can be impacted by deadline misses, task starvation, and race conditions. Missing a hard real-time task can cause a cascade of point failures. If a scheduling algorithm is inefficient, it leads to the possibility that tasks may not be completed on time. Inefficient scheduling algorithms also bring about task starvation, where some tasks do not execute even though they are ready to do so. This can result in a degradation of system performance, as well. The scheduler must also enforce mutual exclusion and synchronization to prevent race conditions. There can also be an overhead if complex algorithms are used (Stallings, 2017).
* iSpace’s LRTOS is susceptible to insufficient algorithm responsiveness. The aerospace environment consists of uncountable hard real-time tasks that, if their deadlines are not met, could result in a catastrophe. Some examples of this could include the LRTOS’ inability to avoid debris efficiently. Space debris poses a constant threat to spacecraft, and the LRTOS must prioritize and execute avoidance maneuvers in real-time to ensure the safety of the spacecraft and its mission. If the scheduling algorithms within the LRTOS are not responsive enough, there may be delays in executing these avoidance maneuvers, increasing the risk of collisions.
* *Mitigation of Resource Monopolization*
* A successful operating system must consider mechanisms to prevent system failure in cases of extreme resource monopolization. Stability and reliability of a system could be compromised if proper mitigation techniques are not in place. One such technique is setting resource limits. The operating system is designed to know which processes can access certain resources, and for how long. Timers can be set to restrict the max time a processor has with a process. The OS can decide how power is distributed among different components. Memory limits can put a restriction on how much random-access memory that a process can use at a given time (Stallings, 2017).
* Priority-based and fair scheduling algorithms can be used to ensure that an operating system drives the correct sequence of events. Allocating resources to less urgent tasks can allow for the retention of finite resources such as power. The OS can decide which events call for priority-based scheduling and which events require fair scheduling. The combination of both provides a balanced approach to dealing with obstacles without risking the integrity of the system. This can also prevent idle time, as tasks with less urgency will get less attention (Stallings, 2017).
* An efficient operating system can collect usage statistics of resources and monitor performance. Such monitoring can include the monitoring of response time. This can help predict future upgrades and the tuning that is needed to be done to improve overall system quality. Monitoring access to resources can allow for an operating system to adjust according to necessary time constraints. In other words, if a process is taking up too much time with a resource or if that resource is sitting idle, the OS can reallocate that resource in real-time (Stallings, 2017).
* The processor can execute instructions from the main memory that has a monitor. Most of the monitor is allocated in main memory and is called the resident monitor. The monitor has a scheduling feature in that a batch of jobs is queued and executed as fast as possible with no idle time. The monitor can also improve job setup time. This is because of job control language, which includes instructions for the monitor (Stallings, 2017).
* Mitigation of resource monopolization is an important aspect for iSpace engineers to consider. The success of a mission depends on a functional unit’s ability to ration its finite resources in outer space and foreign planets. In scenarios where the operating system is hard-pressed to allocate much of its resources to one subcomponent, the OS must have the proper mechanisms in place to ensure this does not result in certain tasks being starved indefinitely.
* An example of this can be seen in the case of a rover collecting samples. The LRTOS can constrain the max CPU time allocated to non-urgent tasks such as routine maintenance while ensuring that much of the processing power is given to sample analyzation and movement of the rover. These allocations can change as mission objectives change over time. For instance, a stationary rover performing a hardware check can make the checking process a priority, while sample analyzation runs in the background. This switching of resource allocation can allow iSpace’s LRTOS to better adapt to mission requirements.
* *Mitigation of Data Loss or Data Corruption*
* Alongside mitigation of scheduling errors, it is imperative for an operating system to consider the possibility of permanent data loss. One way to avoid this is implementing redundancy through means of redundant arrays of independent disks (RAID). Information redundancy introduces fault tolerance by copying and coding data such that errors in bits can be detected and corrected. An example is the error-correction techniques used with RAID disks. RAID uses multiple hard drives to store the exact same data in different places on different disks. For example, RAID 1 replicates data on two separate disks, while RAID 5 disperses information along three or more disks. RAID 1 allows for real-time backup of all necessary data so that if a disk fails, important information is still accessible right away. Using multiple disks allows for multiple ways data can be organized which can then improve overall system reliability. RAID is a set of physical disk drives, but the operating system sees it as just one drive. Information is dispersed along the physical drives through striping. Data is split amongst the disks and any extra space can store extra information which allows recoverability (Stallings, 2017).
* To ensure data preservation, operating systems should proceed with regular data backups. Copies of information should be made constantly, mitigating the risk of data loss due to other failures such as hardware malfunctions. A thread that is only meant to regularly save copies of important information can be made to schedule itself directly with the operating system. Therefore, there is no need for complex coding in the main program to manage data copying. Performing routine backups assists with maintaining integrity of data. Systems can archive copies of data and hold them for long periods of time to access present data in the future (Stallings, 2017). This can allow future developers to refer to relevant data and make the necessary adjustments to make future operations smoother.
* Operating systems can implement the New Technology File System (NTFS) to improve recoverability. NTFS is a file system built on a simple file system model. This system can reconstruct disk volumes and return them to a stable state in the case of a disk failure or system crash. Any changes made to files in this type of system are done with the all-or-nothing principle. In other words, in the case of a crash the change can be undone or finished later. Like RAID, the NTFS stores information in multiple places, allowing redundancy. The NTFS is also efficient at handling large volumes of information within large disks and files. Another feature of the NTFS is journaling, in which a record of all changes is kept (Stallings, 2017). This allows for effective monitoring of updates.
* The NTFS can also handle different kinds of disks with non-standard byte sizes (Stallings, 2017). This can allow for the proper flow of information within the LRTOS. It can also allow compatibility between the LRTOS and different subcomponents. The NTFS integrated with regular backups and RAID, can ensure that the valuable data collected by iSpace can be conserved in the event of a disaster. The overall reliability and resilience of the LRTOS is upgraded. Data recoverability in the event of data corruption is improved, as well.
* In a pressing aerospace environment, it is conceivable that data is prone to corruption. Proper techniques to mitigate the amount of data lost, if any, can be extremely beneficial to iSpace because the integrity of the mission may rely on collected data. This can be seen in the case of a rover collecting samples. Even if the rover is capable of physically collecting the samples, if there is not sufficient time for that data to be stored and analyzed properly, mission efforts become futile. This can be seen in the event of an unforeseen environmental hazard. Recovering lost data can determine if the mission has been a success or a failure. It is crucial that the enterprise sees this risk as a dooming outcome to take the appropriate preventative measures.

* Mitigation of Hardware Failure
* Implementing redundancy tactics in critical components of the operating system can stall the cascading failure of future components. This can involve having a second processing unit, or multiple power supplies. The secondary components can ensure survival of a system in case a single part of it malfunctions. This tactic is considered spatial redundancy where multiple physical components are configured to operate on the same function, or having one component as a backup in case the primary component fails. A range of errors can occur while an operating system is running. The OS must give a response that clears the error condition with minimal impact on running applications (Stallings, 2017).
* An operating system can use traps to handle errors caused by hardware malfunctions. The OS can determine if an error is fatal, and if it is, then the current process is moved to the “Exit” state and a process switch occurs. It can attempt recovery or send a message signaling an error has occurred. Traps can be used as a preventative measure along with regular inspections of hardware (Stallings, 2017). Quality assurance and continuous testing of hardware components is integral to satisfying regulatory requirements. This can also increase the longevity of components, making future operations more feasible. This can also allow for the reuse of certain costly hardware components.
* The use of RAID 6 can be an effective mitigation technique in the case of hardware failure. In this design, two separate parity calculations are stored in different blocks on different disks. In this configuration, data that requires *N* disks always has *N + 2* disks. In this case, even if two disks fail, it is still possible to regenerate data. In other words, three disks must fail within the mean time to repair interval to cause irreversible data loss. Setting a protocol used for communication between different hardware components serves to reduce the likelihood of system failure. A feature of protocols are semantics, which include control information for coordination of error handling. An effective protocol architecture enlists a structured set of modules involved in communication between subsystems (Stallings, 2017). This can serve to input failsafe mechanisms that can prevent multiple point failures in hardware.
* iSpace related hardware is thoroughly checked for errors using a checklist that contains extensive requirements and standards these pieces of hardware must meet. This checklist corresponds to the guidelines laid out by the FAA (Mckenna, 2013). Hardware failure in iSpace’s LRTOS can have severe implications, especially if this occurs in the process of a mission at multiple points of a system in outer space. iSpace utilizes titanium in its heat shields to protect components of the OS in the extreme environments of outer space. High temperatures are used to mold the titanium into the necessary shapes (Hefti, 2010).
* The mitigation of hardware failures is imperative for iSpace to drive the success of its missions. In challenging environments of outer space and distant planets routine checks, spatial redundancies, and features such as RAID 6 will help iSpace’s LRTOS during an objective. Identifying potential hardware issues before they result in catastrophic failures is crucial for iSpace engineers. In the case of a spacecraft’s battery-related failure, it is important that engineers include at least two different fail safes of each critical component. This can also be seen in the case of a rover collecting samples where the navigation component and the sample-analyzing component should both have multiple battery redundancies.
* *Mitigation of Network Failure or Data Breach*
* Firewalls are an effective way to improve network security and stop unauthorized access to systems. Firewalls can protect networks of systems from network-based security threats. The firewall has specialized security measures built within it involving protection of sensitive files. In firewalls implemented into hardware and software, all traffic moving in and out must pass through the firewall. This is done by having the firewall act as a gate. The firewall is designed to know what information is considered authorized traffic through the implementation of security policies. These security policies contain rules to help the firewall identify the characteristics of what is considered authorized traffic (Stallings, 2017).
* Intrusion detection systems (IDSs) are security services that monitor and analyze system events to find and give real-time notice of an unauthorized attempt to access system resources. The two types of IDSs are a host-based IDS and a network-based IDS. A host-based IDS observes the characteristics and events that happen within one host, monitoring for suspicious activity. A network-based IDS monitors network traffic and analyzes transportation to find suspicious activity. Three components involved with IDSs are sensors, analyzers, and the user interface. Sensors collect data and forward information to the analyzer which determines if an intrusion has occurred. Any output from this component means that an intrusion has occurred, and that output can provide evidence of the intrusion (Stallings, 2017).
* Compile-Time defenses can be used to defend against an attack that takes advantage of buffer overflow. They do this by manipulating programs during compilation. Measures to achieve compile-time defenses can include choosing a complex language that does not allow buffer overflows. Compile-time defenses also include using safe standard libraries or adding code that detects corruption of stack frames. Runtime defenses attempt to find and abort attacks in executing programs. One way to defend against a buffer overflow attack is to block execution of malicious code on a stack. Another runtime mitigation technique can be using address space randomization in which the location of the stack address is changed for each process (Stallings, 2017).
* iSpace is a target for intruders and competing enterprises. Designers must implement the best tactics to thwart any attempts made to corrupt aspects of the LRTOS. Any malicious attempts to manipulate the functional units and data collection systems of iSpace could result in surmountable losses. Firewalls can serve as the first defense mechanism for iSpace systems. In both hardware and software, the enterprise can control and monitor the flow of traffic through networks. The valuable data that is collected by units such as rovers can be protected against data breaches. Because IDSs provide real-time monitoring and detection of unauthorized access to system resources, they are an important defense mechanism for iSpace. Accurate and up-to-date notifications by an intrusion detection system can prevent irreplaceable information from being lost or stolen by a third party (Stallings, 2017).
* Data breach mitigation within the LRTOS involves countering thread from malicious software. Malicious software can invade system defenses and execute on those systems or other target systems. If this were to happen during a mission, the integrity of iSpace functional units would be jeopardized. As one more precautionary measure, employees of iSpace are given role-based access control. This access depends on the identity of the user and the exact role that user can take for a set of tasks (Stallings, 2017). These mitigation tactics can be the saving grace of operations involving aerospace. Intrusions within iSpace cannot be allowed to go unnoticed.
* *Mitigation of Improper Task Scheduling & Prioritization*
* Aerospace conditions are inherently unpredictable. Constantly changing variables include trajectories of debris, unexpected system failures, or variations in environmental conditions. Therefore, iSpace must account for as many external variables as possible in its scheduling algorithms. These algorithms must be responsive, adaptive, and capable of adjusting task priorities based on changing environmental conditions. In a complex system like iSpace's LRTOS, multiple scheduling algorithms are implemented to deal with different types of tasks or components. It is essential that these algorithms work in a synchronized manner with each other to avoid conflicts or interferences that disrupt mission objectives. For example, the scheduling algorithm responsible for communication tasks should not interfere with the algorithm involved in navigation tasks.
* Bitmap schedulers can be incorporated into operating systems to support and track multiple priority levels. This ensures timely execution of processes and makes sure no lower priority process blocks a higher one. In a bitmap scheduler, only one thread can exist at a priority level at a time. When a running thread is preempted, the next thread with the highest priority that is ready will be executed. Because only one thread exists at a priority level, the scheduler has less decisions to make regarding what threads at each level should be executed next (Stallings, 2017).
* Multilevel queue schedulers are like bitmap schedulers in that they support up to 32 levels of priority. The MLQS allows multiple threads in priority levels, but they are restrained by available system resources. When a thread is suspended, the scheduler determines what thread to execute next based on whether the next thread is on the same priority level as the suspended one. If it is, the scheduler executes the thread in the front of the queue. If it is not, the scheduler moves down a priority level in search of a thread that is in the ready state. The MLQS can also incorporate time quanta. If threads are waiting for a running thread in the same priority level, after a certain time the scheduler will block the running thread and execute the next one in line. Bitmap schedulers and multilevel queue scheduling can help mitigate risks associated with improper task scheduling by managing the interactions between real-time processes and routine processes (Stallings, 2017).
* To support the urgency of decision making in space, processes can be divided up into sections of importance. Real time processes are at the highest priority levels and are guaranteed to be selected to run before any kernel or time-sharing processes. Real-time processes can utilize preemption to override kernel processes and user processes. Kernel processes are selected to run before time-sharing processes, but after real-time processes. Time-shared processes are of the lowest urgency and are run last if needed (Stallings, 2017). Examples of this can be the need to parcel processes in case of emergency. Navigation processes must be given top priority in case of debris collision risk. This can allow the spacecraft to maintain lower-level priority tasks such as routine maintenance checks while executing the proper evasive maneuvers when needed.
* Since iSpace endeavors are often high stakes, the LRTOS’ ability to correctly prioritize tasks is imperative for mission success and safety. In emergency situations, the algorithm setup up of incorporating a round-robin feature of time quanta within priority levels is an appropriate method of ensuring mitigation of disaster (Stallings, 2017). iSpace's missions in space involve time sensitive and high-risk tasks, making it crucial for the LRTOS to prioritize tasks efficiently. This insinuates the management of various space mission functions, such as navigation of a spacecraft or processing collected data, by deciding what functions require most urgency and letting those execute first. like navigating the spacecraft or collecting scientific data, by determining which tasks are most urgent and handling them first. This ability is especially important during cases of emergency where a rover must quickly navigate dangerous terrain.
* The LRTOS utilizes an effective combination of scheduling techniques to make sure tasks are handled in the right order and at the right time. One of these techniques includes a round-robin system that ensures all tasks of the same importance level get equal attention without any single task taking over. This is critical during intense moments of a mission where tasks like navigation need to be managed alongside other important tasks without any delays. This in combination with bitmap schedulers and MLQS allow iSpace functional units to swiftly adapt in dynamic environments, enabling them to be responsive to emergencies while managing routine operations simultaneously.

## Future Considerations Using Emerging Technology

* *Multiplanetary Expansion & Mission Planning – Integration of Artificial Intelligence*
* iSpace is working toward building bases across select planets of the solar system, enabling interplanetary communication. iSpace would benefit from the implementation of artificial intelligence in its spacecraft and rovers to increase the probability of mission success and system reliability. iSpace also requires A.I. to intricately design the optimal blueprints for future space systems that will span across the solar system. It is impractical to send an astronaut to the deep stretches of our solar system due to the time it would take and difficulties in communication. However, if A.I. is integrated into the LRTOS of iSpace entities, it could allow for longer mission duration while reducing the need for human intervention. This could also allow iSpace to maximize its distance covered by creating communication and recovery checkpoints throughout the solar system. For example, it is not feasible to send an astronaut directly from Earth to Pluto, but one can imagine the possibility of creating pit stops along the way to replenish resources (Qiu, 2024).
* Artificial Intelligence enables autonomous decision-making and problem-solving capabilities within its spacecraft and rovers. In cases of communication latencies, A.I. can appropriately respond to external variables such as debris. Artificial intelligence can also optimize the use of resources in real-time by analyzing data and predicting efficient allocation of resources. This can greatly extend mission duration and conserve finite resources. iSpace also benefits through artificial intelligence’s ability to plan missions by assessing environmental conditions and available resources. The impact A.I. can have on iSpace’s functional units is nothing short of grand. Active adaptation of spacecraft in complex missions and harsh environments can be improved by implementing artificial intelligence (Qiu, 2024).
* *Astronaut Training and Spacecraft Simulation – Introduction of Virtual Reality*
* If iSpace wishes to be the facilitator of turning humans into a multiplanetary species, it must consider how its astronauts will prepare for dynamic environments. The current training methods iSpace employs are outdated. It uses projectors and 3D glasses as its simulation technique (Pirker, 2024). If the company wishes to keep up with its competitors, it must upgrade its simulation tactics. As technology advances, its operators must keep up with the changes. Virtual reality introduces a way for iSpace employees to visualize the rapidly changing environment they will be faced with. iSpace needs accurate virtual replication of real environments for training purposes. Astronauts must learn how to utilize different tools while evaluating cases of emergencies.
* iSpace aims to cut costs. Training for missions without virtual reality is often costly and takes much time to set up. Virtual reality can allow for more frequent training sessions. It is difficult to fully replicate the scenario an astronaut will face without the use of virtual reality. iSpace also requires a more involved and realistic method of controlling its rovers during planetary exploration. An operator is provided with very little feedback and immersion without having the enhanced 3D situational awareness provided by virtual reality. Astronauts must be properly trained in environments representing the destination (Pirker, 2024). Current measures of iSpace utilize swimming pools to imitate the environment of outer space.
* *Cutting Costs and Optimizing Components – Applications of Additive Manufacturing*
* If iSpace plans to expand its horizons, it must consider methods to reduce production costs of future products or previously degraded components. Replacing worn out components is becoming a costly objective for iSpace. It is difficult for engineers to perfectly construct complex geometry within its structures because this process requires immense accuracy and precision. iSpace also wishes to explore alternative, more sophisticated architecture designs of its spacecraft and operating system components. Engineers cannot afford to do this multiple times without the use of 3D printing (Lim, 2016).
* iSpace also aims to improve passenger experience and increase overall safety while reducing mission durations. If the use of 3D printing can cut costs enough, iSpace can deploy multiple spacecraft, rovers, and satellites at the same time. This would inevitably support any time constraints set by celestial bodies that mission planners must account for. Currently, iSpace mission planners are largely at the mercy of physical forces such as gravity. The enterprise wishes to upgrade its hardware components using cutting-edge 3D printing. Inserting these upgraded hardware components may reduce response time by the system (Lim, 2016).
* *Artificial Intelligence*
* AI can be categorized into analytical, human-inspired, and humanized AI based on its behavior. It can display cognitive behavior, understanding of emotions, and social awareness. Artificial intelligence stems from computer science with an ability to understand natural language, recognize pattern repetition, learn from past encounters, and solve problems. These qualities enable AI algorithms to guide humans through complex decision making, especially in large corporations. It can also be categorized by its stage of evolution in terms of artificial narrow, general, or super intelligence. Artificial intelligence systems have become common in many areas of math and science. AI is considered a science that studies theory, methodology, and technology to multiply human capabilities. It excels at decision making and reinforcement learning. Due to its ability to learn constantly, AI can dissect complex information such as mental health issues in humans. By taking in vast amounts of information, a system can properly assess the occurrence of a mental health condition (Mueller, 2018).
* Engrossing the field of computer science, AI is constantly working on making algorithms and systems that imitate human intelligence to carry out various jobs. A continued aspect of interest regarding AI is testing theories related to human-like behavior and consciousness. Developers’ goals are to create algorithms that can simulate actual cognition close to a human. With these capabilities, an effective AI system can communicate with its surroundings as if it were a person. It can make predictions and mimic emotions. Creating algorithms that can “learn” from data to make predictions is crucial in an AI system’s ability to adapt to future circumstances and make decisions autonomously (Mueller, 2018).
* Machine learning is the study of programs that autonomously improve their ability to carry out tasks without human intervention. Unsupervised learning methods dissect data and find patterns to make predictions without any other intervention involved. Supervised learning involves training a model on labeled data. Reinforcement learning can be comparable to training a pet where better decisions are made over time due to trial and error. Natural language processing enables programs to have the ability to read, write, and execute in human languages (Mueller, 2018).
* Virtual Reality
* VR is the simulation of virtual experiences where a human can immerse themselves into the virtual environment in real-time. Virtual reality has become more affordable and accessible in the past ten years, leading to innovation in many fields of science. VR extends its capabilities far more than simply for entertainment purposes. The interactive qualities presented by this technology allow for realistic training experiences. Virtual reality can be used in medical, educational, or design applications. VR allows immersion into simulations that would normally be far too dangerous and costly to implement. VR also increases safety that would normally be threatened due to difficult-to-implement training scenarios (Pirker, 2024).
* One can think of VR as a computer-generated environment that represents a real-world environment but doesn’t exist. Virtual reality contains subgenres that have different characteristics. One of which is augmented reality which differs from VR because it adds computer elements to the physical environment instead of creating a whole simulated world from scratch. It can be thought of as looking through glasses that overlay images over certain objects. Mixed reality contains characteristics of augmented and virtual reality. This allows for human interaction between the overlaid digital environment and the physical environment itself (Mealy, 2018).
* Augmented virtuality is known as merged reality, and it is the opposite of augmented reality. In augmented virtuality instead of a digital component being overlaid on a physical environment, a physical component is overlaid into a digital environment. This can allow something such as human hands to know how to move in certain ways given a specific environment. Augmented reality is constantly improving its motion tracking techniques. Experiments can be interactively conducted in these types of environments. (Mealy, 2018).
* Additive Manufacturing
* Additive manufacturing is also known as 3D printing. It involves making an object from a model of three-dimensional data by joining materials together one layer at a time. Some of these layer stacking methods include extrusion, sintering, and melting. Normal manufacturing involves subtractive techniques, but additive manufacturing does the opposite by adding materials one layer at a time. The cross section of the 3-D model is used as a template for construction. This type of additive manufacturing reduces the need for adjustments after the fact, eliminating the use of cutting tools. Computer-aided design (CAD) tools help optimize the overall design of what is being manufactured. This can increase customization of any part necessary (Getachew, 2023).
* Additive manufacturing has revolutionized many industries. 3D printing allows for lighter systems with sound-absorbing abilities. This has opened many advancements in technology. Additive manufacturing is effective because it combines different materials into one compact design. This upgrades functionality and enhances the performance of a system. The nesting feature adopted by 3D printing is the optimization of material organization. This increases compatibility of materials by placing materials with similar jobs close together. By changing the internal structure of one material, that material can display behaviors equivalent to a structure with multiple materials inside of it. This in conjunction with nesting increases overall efficiency in any system the components are used in (Getachew, 2023).
* The first step of 3D printing is design of the object, which can be done using software such as computer-aided design. This model can be adjusted accordingly based on criteria laid out by the designers. Once the model is designed, it is formatted into an understandable format such as stereolithography (STL). An STL is a triangulated representation of the model that has been designed in CAD. After formatting, the 3D model is sliced into thin layers. Once the printer is calibrated, the object is printed and processed. Postprocessing can involve the removal of support structures required for the actual printing process, but not the final product (Getachew, 2023).
* *Benefits of Artificial Intelligence for iSpace*
* The emergence of advanced artificial intelligence has opened many doors for the aerospace industry. The ever-evolving nature of AI allows for constant developments and upgrades in aerospace systems. With high-performance chips, AI can help with various aspects of iSpace. It can help with the intelligent design and management of iSpace systems, and real-time adaptability of iSpace spacecraft and rovers in hazardous environments. AI can also lead to extending exploration even deeper into space. Because AI can learn from prior experiences, it can assist with instances such as a rover faced with an obstacle collecting samples. If the intelligent system has come across a similar event in the past, it can come up with a better solution quickly (Qiu, 2024).
* AI enables iSpace spacecraft to autonomously perceive information from its surroundings and make decisions such as navigation. Integrating artificial intelligence with the LRTOS of iSpace can support the longevity of spacecraft and rovers. Through the techniques of deep and reinforcement learning, functional units of iSpace will be better equipped for unpredictable environment conditions. AI can help improve mission efficiency and reliability by automating tasks like fault management and mission planning. Intelligent design of space systems refers to using AI for optimization, analyzation, and layout of system architecture. It involves the use of smart materials which give structural weight reductions and optimize components. Smart materials can be developed to protect against space debris and monitor repairs (Qiu, 2024).
* *Benefits of Virtual Reality for iSpace*
* iSpace can benefit from great innovations brought by virtual reality. iSpace can benefit from implementing virtual reality mechanisms into its training programs. The simulation of environments can help astronauts become acquainted with physical constraints. Feedback mechanisms can be used to give the user a more immersive experience. This allows for an accurate representation of where the trainee will be. Implementation of virtual reality can also help mission planners in sequencing their maneuvers more effectively. Engineers of iSpace will benefit from cost reduction, as well. The physical creation of environments is time consuming. VR also allows for precise remote operations, and abstraction of complex information. This can aid in upgrading software, and manufacturer planning (Pirker, 2024).
* Virtual reality is flexible in that it can be modified to fit various applications. It can simulate a real-time environment without the actual consequences of disaster. This can help astronauts manage their emotions of stress and fear. An example can be seen of simulating a space station for astronauts who will be onboard performing maintenance routines. This would allow the company to consider risks and mitigations of previously unknown hazards. VR can also enhance remote operations using an immersive environment. For example, if an astronaut must remotely operate a rover, the operator has wider viewing capabilities and less operating restrictions than normal video streams (Pirker, 2024). Another example of proper training can be seen if an operator is controlling a rover’s arms and is provided with negative feedback. This can have the operator make the decision to adjust movements accordingly.
* *Benefits of Additive Manufacturing for iSpace*
* Additive manufacturing can allow engineers of iSpace to work with near perfect, state-of-the-art components. Each of these components can be seamlessly integrated with one another by nesting. The aerodynamic requirements in high pressure environments can be satisfied using 3D printing. iSpace can ensure safety, and possibly reduce mission duration due to the cutting-edge aerodynamics provided by additive manufacturing. iSpace engineers can also benefit from eliminating the need for multiple processing steps in normal manufacturing methods. Engineers can design their intricate components, true to their model requirements, in a single step (Getachew, 2023).
* 3D printing allows for increased shape complexity. This enables iSpace to be more imaginative with its design process and stray from conventional blueprints. The following are different types of additive manufacturing methods, each with their own set of advantages: fused deposition modeling, selective laser sintering, and direct metal laser sintering. Fused deposition modeling is a layer-by-layer deposition of melting thermoplastic used widely in the aerospace industry due to its ability to create strong and lightweight structures. The high accuracy of these methods and the strength of their materials produced ensures safety and integrity of iSpace functional units. 3D printing also allows iSpace to constantly modify its designs, and remake components in instances of failure or degradation. Additive manufacturing can upgrade resistance to extreme temperatures and pressures (Getachew, 2023).

1. References

Amdahl, G. M., Blaauw, G. A., & Brooks, F. P. (2000). Architecture of the IBM System/360. *IBM Journal of Research and Development*, *44*(1.2), 21–36. <https://doi.org/10.1147/rd.441.0021>

Campbell-Kelly, M., & Garcia-Swartz, D. (2015). From mainframes to smartphones: a history of the international computer industry. *Choice Reviews Online*, *53*(02), 53-085553-0855. <https://doi.org/10.5860/choice.192962>

Comins, N. F. (2017). The Traveler’s Guide to Space: for One-way Settlers and Round-Trip Tourists. Columbia University Press.

Dulaney, E. (2010). Linux All-in-One for Dummies. John Wiley & Sons.

Fanfakh, A. (2016). Energy Consumption Reduction for Asynchronous Message-Passing Applications. The Journal of Supercomputing, 73(6), 2369–2401. <https://doi.org/10.1007/s11227-016-1926-1>

Getachew, M. (2023). The Current State of the Art and Advancements, Challenges, and Future of Additive Manufacturing in Aerospace Applications. *Proquest*. https://doi.org/10.1155/2023/8817006

Hefti, L. (2010). Fabrication of Titanium Aerospace Hardware using Elevated Temperature Forming Processes. SAE International Journal of Aerospace, 3(1), 173–179. https://doi.org/10.4271/2010-01-1834

Jenkins, D. (2001, April 5). *Advanced Vehicle Automation and Computers Aboard the Shuttle*. Www.nasa.gov. <https://www.nasa.gov/history/sts1/pages/computer.html>

Karlsson, S., & Mats Brorsson. (2005). Priority Based Messaging for Software Distributed Shared Memory. Cluster Computing, 6(2). https://doi.org/10.1109/ipdps.2001.925152

Lockney, D. (n.d.). *Search | NASA Software Catalog*. Software.nasa.gov. <https://software.nasa.gov/software/category/aeronautics>

McKenna, E. (2013, April 1). *Real-Time Operating Systems*. Aviation Today. https://www.aviationtoday.com/2013/04/01/real-time-operating-systems/

Moyer, M. (n.d.). *Common Standard Billing Unit (SBU) - HECC Knowledge Base*. Www.nas.nasa.gov. <https://www.nas.nasa.gov/hecc/support/kb/common-standard-billing-unit-(sbu)_271.html>

Mealy, P. (2018). *Virtual & augmented reality for dummies*. For Dummies, A Wiley Brand.

Mueller, J. (2018). *Artificial intelligence for dummies*. John Wiley & Sons, Inc.

Mukherjee, A. (2015). Learning Boost C++ Libraries. Packt Publishing Ltd.

Murata, T. (2003). On Shouting “Fire!”: Regulating Decoupled Communication in Distributed Systems. Middleware ’03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, 222–241.

Pirker, J. (2024). *The Potential of Virtual Reality for Aerospace Applications*. IEEE Xplore. <https://ieeexplore-ieee-org.coloradotech.idm.oclc.org/document/9843324>

Qiu, Z. (2024). *Applications and Challenges of Artificial Intelligence in Aerospace Engineering*. IEEE Xplore. https://ieeexplore-ieee-org.coloradotech.idm.oclc.org/document/10206205

Stallings, W. (2017). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson Education (US). <https://coloradotech.vitalsource.com/books/9780134700113>

Ying, Z., Feng, C., & Zhao, Z. (2020). Electronic-photonic arithmetic logic unit for high-speed computing. *Nature Communications*, *11*(1). <https://doi.org/10.1038/s41467-020-16057-3>

Zhou, Z. (2010, August 2). Spacecraft Attitude Tracking and Maneuver Using Combined Magnetic Actuators. *AIAA Guidance, Navigation and Control Conference*.