**Architecture of PintOS**

**21065719 Bit Na Yoon**

**22027036 Farhan Zakir**

**21003020 Eyad Alshehri**

PintOS is structured around the most basic feature, threads, with additions of user programs, virtual memory, and file systems.

**- Process state**

A process is a unit that represents a program in execution within an operating system, and inside each process, there is at least one thread which is in fact the unit that is scheduled. Each process is executed in an independent memory space that it has been allocated, and the operating system manages the creation, control, and termination of processes, as well as interactions between multiple processes.

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

The process has these stages as illustrated. Initially, when a process is created, it is in a state where it is not yet ready to be executed. The process awaits CPU execution in the Ready stage; all execution conditions are satisfied, but the CPU has not yet been allocated, so it can be said to be in a waiting state. In the Running stage, the process is executed on the CPU, and in the Blocked stage, the process waits for the occurrence of a specific event, such as waiting for I/O completion or waiting to acquire a lock. In this state, the process does not use the CPU. In the Terminated stage, the process has either completed execution or has been terminated for some reason. This is determined by the process scheduler, for instance, the scheduler can select one of the processes in the Ready state and change it to the Running state. If a process in the Running state requests an I/O operation or its time allocation ends, the scheduler reclaims the CPU and reverts the process state to Blocked or Ready.

**- Thread**

A thread is a unit of execution within a process that has an independent execution path as part of the process and can run in parallel or concurrently by sharing the resources of the process. A thread shares the Code, Data, and Heap areas within a process, but has its own separate stack. It is a flow of multiple executions operating within a process, executing while sharing address spaces or resources within the same process among threads. Multiple threads within the same process share the same Heap space, but in contrast, a process cannot directly access the memory of another process. Each thread has separate registers and stacks, but they can read and write to the Heap memory. In PintOS, a multithreading model is used, which allows for fast response times and saves memory resources because all memory excluding the Stack area is shared among threads. It can be considered advantageous that each thread uses less memory and resources than independent processes. However, multithreading can lead to the termination of all processes if one thread spoils the resources within the process because the process shares the same memory space. Inevitably, synchronization issues can occur due to resource sharing.

The stack of a thread is created when the thread itself is created, with each thread possessing its own independent stack space. This implemented in the thread\_create function within the threads/thread.c file, where a new thread structure is created and initial values are set for the thread's stack pointer and stack pages. There's a problem that can occur in the stack known as stack overflow, which happens when there are too many function calls and the stack space is exhausted. To solve this problem, the thread can allocate additional stack pages as needed to expand the stack space, allowing each thread to run stably.

**- User program**

User programs include processes such as argument passing, user memory management, system calls, process termination, and denying writes to executables.

A system call is a method used by a user program that requires the support of the OS, and before trapping into kernel mode, it needs to confirm whether the incoming argument values for the syscall are valid addresses, and whether they are not attempting to make fatal changes to the system.

1. Argument passing : This is a method of providing data to functions or processes, where the argument is stored in the kernel stack, and the process of passing the argument to the function or process is controlled. Each process must create the illusion that the CPU operates only for that process, but in reality, it doesn't. That is, each process must be setting memory, scheduling, and other states.  
2. User memory : When a stack pointer is given an incorrect address, it can point to an unexpected different memory area. If this happens, data can be written to the wrong location, so this process prevents access to incorrect memory.  
3. System calls : This process allows programs to request services from the kernel using the system call interface.  
4. Process termination messages : Through the system call, the process is terminated using a function called 'exit'. This process must ensure that when the process execution ends, all resources used by that process are returned to the operating system.  
5. Denying writes to executables : This feature prevents changes to the executing code and blocks write operations to the executing file to prevent the executing process from being unexpectedly changed.

**- Virtual memory**

Virtual memory is one of the management techniques through which the operating system abstracts three main areas of the computer or machine: CPU, file, and memory. Each process is allocated one virtual memory, and multiple threads running within a process share this single virtual memory space. This approach is commonly used in multitasking operating systems and serves as a method to provide a memory area larger than the actual physical memory.

In PintOS, the paging technique is utilized, where virtual memory is managed using fixed-size blocks called pages. A page represents a partition of virtual memory with a fixed size. Correspondingly, frames are blocks of physical memory that are also divided into the same size as pages, commonly set to 4096 bytes (4KB) in PintOS.

.**텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

The entire executable file is currently being loaded into memory from the start in pintOS. However, this is very inefficient because even the unexecuted parts are loaded into memory, which needs to be addressed.

- Demand Paging : Instead of loading all memory into physical memory when a program runs, only some pages needed initially are loaded into memory. This makes it efficient as only parts of the program are always executed, not the whole. The remaining pages are loaded as needed, which is handled through page faults.  
**-** Page Fault: In Demand Paging, it occurs when the page that the process is trying to access is not in physical memory. If a page fault occurs, the OS finds the necessary page from the disk, loads it into physical memory, and allows the process to continue running. However, if it occurs frequently, a lot of time is spent loading pages on the system, so the number of occurrences should be minimized.

텍스트, 스크린샷, 번호, 도표이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

In the existing memory allocation method, a blank space is left between the Heap and Stack areas to allow for expansion if the Heap or Stack is insufficient. However, if there are many processes that do not need to expand the Heap and Stack, there will be many spaces allocated but not used, resulting in memory waste.

Therefore, placing Segments in this way can reduce unused space and use memory efficiently.

**- File system**

File system is a system that organizes and stores files or data in a computer, enabling easy discovery and access to them. The primary roles of a file system include:  
- File Allocation: Allocating new files to the disk.  
- File Access: Providing methods for users to access desired files from the disk.  
- File Protection: Managing access permissions for files.  
- File Consistency: Ensuring that the contents of files remain intact and uncorrupted.

File operations involve actions such as Create, Write, Read, Delete, and more, and the operating system (OS) must provide system calls to handle these operations.

In the context of PintOS, it is currently in a state where it can only process I/O. The implementation for reading, writing, and managing files is not present within PintOS. Therefore, the objective is to make the thread aware of the files it uses, allowing necessary user programs related to the file system to be executed. Then, the goal is to implement handlers to handle each system call that arises while executing these user programs.