Review

- OS as a Resources Manager
  - Process Management
  - Memory Management
  - File Management
  - Input/Output System Management
  - Deadlock Management
  - Cache Management
- Operating System Structures
  - Monolithic
  - Layered System
  - Microkernels
  - Virtual machine
  - Client-Server module
  - Exokernels

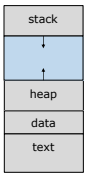
Preview

- Processes
- Process Model
- Process Creation
- Process Termination
- Process States
- Process Table (Process Control Block)
- Process with Multiple-Threads
- Process Scheduling
  - Scheduling Queues
  - CPU Scheduling
  - Context Switch
- Process Creation in Linux
- Process Termination in Linux
- Android Process Hierarchy

The Process

- A program becomes a process when an executable file is loaded into memory. A process is a program in execution.
- When a program start to run, OS keep its status in a process table (process control block) until its termination.
- The memory layout of a process is typically divided into four sections.
  - Text section—the executable code (fixed)
  - Data section—global and static variables, (fixed)
  - Heap — used for dynamic memory allocation (change size during execution)
  - Stack section—temporary data storage for local variables when a function is called (function parameters, return addresses, and local variables) (change size during execution)

The Process



Layout of a process in memory

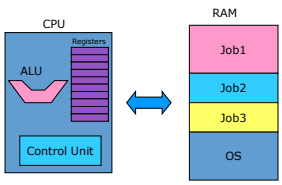
- Text – program code
- Data – global variables
- Stack – local variables for function call
- Heap – dynamic memory allocation

the stack and heap sections grow toward one another, the operating system must ensure they do not overlap one another

The Process

- Just now we assume that there is only one CPU (with single core) in our system.
- Multiprogramming – several jobs are loaded in a memory, operating system simulate Pseudo parallelism (virtual).
- OS schedule the CPU times for processes by switches from one process to another based on the scheduling algorithm and process state (by short-term scheduler).

The Processes



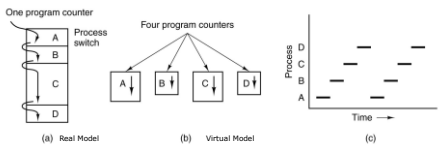
The Processes

The Process Model

We can consider processes as two different point of view.

- **Real Model** – Multiprogramming (processes are sharing resources)
- **Conceptual (virtual) Model** – each process has its own virtual CPU (ALU, PC, registers, stack pointer) and RAM
- Check the following concept
  - Virtual machine-each OS run on its own machine
  - Virtual memory – each process has its own Memory
  - Virtual process model – each process run on it's own machine

The Process Model



The Process Model

- With the CPU switching back and forth among the processes, a process can hold CPU during its time term.
- The time term which assigned to each processes might not be uniform.
- The time term for a process might need be calculated based on the types of jobs – I/O bounded, CPU bounded or priority (usually CPU bounded job need more CPU time then I/O bounded job).

Process Creation

Process Creation

- System initialization - When an OS is booted, several processes (or threads) are created and run concurrently.
- A running process create a process by system call– running process request a system call to create one or more new processes – web server example
- A system user create by executing a program – in interactive system, users can start a program by typing a command or click icon
- Initiation of a batch job – mainframe computer

Process Creation

- UNIX – system call fork create a new process
- In Window – system call CreateProcess create a new process
- Once a child process is created, both parent and child have their own distinct address – two process might need inter-process communication (for sharing resources)

Process Termination

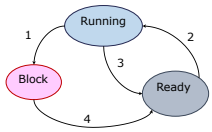
A process terminate due to one of the following condition

- Normal exit – a process finish its job (voluntary)
- Error exit – the process itself discovers a fatal error – compiler try to compile a program, there is no such a file (voluntary).
- Fatal error – error caused by the process– a process try to modify the memory location where other process is located (involuntary)
- Killed by another process – when a deadlock is discovered, OS terminate a process at a time to resolve the deadlock (involuntary)

Process States

- A process must be in one of state
  - Running State – a process is currently being executed by using CPU.
  - Block State (or Waiting State) - The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
  - Ready State – a process is ready to use a CPU but it is not currently available.

Process State



1. Process blocks for input
2. Short-Term Scheduler picks a process since CPU become available
3. A process time out its time term
4. Input becomes available

Process Table (Process Control Block)

- When a process is created, OS stores its run time information in a process table (Process Control Block).
- Process table contains information associated with a process.
  - Process State – ready, running or blocked
  - Program Counter – address of next instruction
  - Contents of CPU registers – Snapshot of CPU
  - CPU scheduling information –priority, pointer to scheduling queue,…
  - Memory management information –page table, segment table,.. Depends on memory management method
  - Accounting information – the amount of CPU time used, CPU time limit, job number,…
  - I/O status information – list of I/O devices allocated, list of open files,…

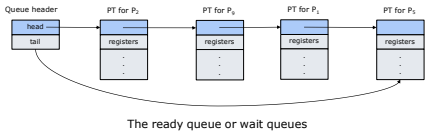
Process with Multiple-Threads

- Most modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time.
- This feature is especially beneficial on multicore systems, where multiple threads can run in parallel.
- A multithreaded word processor could, for example, assign one thread to manage user input while another thread runs the spell checker.
- On systems that support threads, the process table is expanded to include information for each thread.

Process Scheduling
(Scheduling Queues)

- The objective of multiprogramming is to maximize CPU utilization.
- When a CPU become available, short term scheduler (CPU scheduler) select a process from the ready queue and let it use CPU.
- There are two types of queues which holds pointers to process tables for processes in block state and ready state.
  - Ready Queue – holds pointers to process tables for processes in ready state.
  - Wait Queues - holds pointers to process tables for processes waiting for some events(I/O devices, child termination, interrupt, semaphores).
- This queue is generally stored as a linked list; header contains pointers to the first process table in the list, and each process table includes a pointer field that points to the next process table in the queue.

Process Scheduling
(Scheduling Queues)

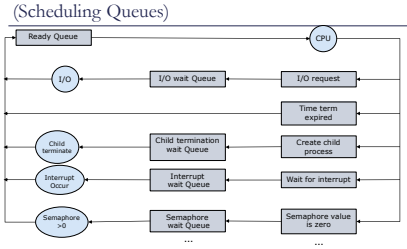


The ready queue or wait queues

Process Scheduling
(Scheduling Queues)

- Once the process is allocated in a CPU and is executing, one of several events could occur:
  - The process could issue an I/O request and then be placed in an I/O wait queue.
  - The process could create a new child process and then be placed in a wait queue while it awaits the child's termination.
  - The process could be removed forcibly from the core, as a result of an interrupt (preemptive scheduler) or having its time slice expire, and be put back in the ready queue.
  - The process tries down semaphore but semaphore value is 0, and be placed in a wait queue for the semaphore.
  - …

Process Scheduling
(Scheduling Queues)



Queueing-diagram of process scheduling

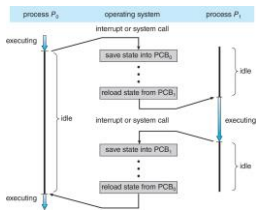
Process Scheduling
(CPU Scheduling)

- A process migrates to the ready queue from various wait queues throughout its lifetime.
- The role of the short-term scheduler (the CPU scheduler) is to select a process that are in the ready queue and allocate an available CPU.
- The short-term scheduler select a process from ready queue base on rules.
  - Shortest Job First
  - Shortest Remaining Time
  - Round Robin
  - Priority Queue
  - Guaranteed Scheduling
  - Lottery Scheduling
  - …

Process Scheduling
(Context Switch)

- Interrupts cause the operating system to change a CPU from its current task and to run a kernel routine by context switch.
- Context switch - performing a state save of the current process to its process table and a state load of a process from its process table.
- Context switch time is pure overhead (CPU must be idle during context switch), since the system does no useful work while switching.

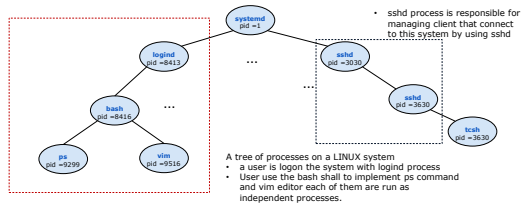
Process Scheduling
(Context Switch)



Process Creation

- A process may create several new processes. Each of these new processes may create other processes, forming a tree of processes.
- Most operating systems (including UNIX, Linux, and Windows) identify processes according to a unique process id (or pid), which is typically an integer number.
- In Linux, once the system has booted, the **systemd** (**init** in UNIX) process is created with pid = 1 and it becomes root parent process of all processes.
- The systemd process creates processes which provide additional services such as a web or print server, an ssh server, and so on

Process Creation



- sshd process is responsible for managing client that connect to this system by using sshd

A tree of processes on a LINUX system
- a user is logon the system with logind process
- User use the bash shall to implement ps command and vim editor each of them are run as independent processes.

Process Creation

- Linux systems provide the pstree command, which displays a tree of all processes in the system. Command pstree with –p flag show process tree with pid

  pstree –p
- When a process creates a child process, that child process need certain resources (CPU time, memory, files, I/O devices) to accomplish its task.
- A child process may be able to obtain its resources directly from the operating system, or it may be constrained to a subset of the resources of the parent process.
- The parent may have to partition its resources among its children, or it may be able to share some resources (such as memory or files) among several of its children.

Process Creation

- When a process creates a new process, two possibilities for execution exist.
  1. The parent continues to execute concurrently with its children by waitpid() system call with option.
  2. The parent waits until some or all of its children have terminated by wait() system call.
- There are also two address-space possibilities for the new process:
  1. The child process is a duplicate of the parent process (it has the same program and data as the parent).
  2. The child process has a new program loaded (with exec) into it and run.

Process termination

- A process terminates by calling the exit() system call. At that point, the process may return a status value (typically an integer) to its waiting parent process (via the wait() system call).
- All the resources of the process are deallocated and reclaimed by the operating system.
- A parent process may possibly terminate one of its children by using signal with signal hander for several reason.
  - The child has exceeded its usage of some of the resources that it has been allocated (when parent and child run currently, parent send signal to a child process to terminate it)
  - In some system, the parent is exiting and the OS does not allow a child to continue if its parent terminates.

Process Termination

- Zombie process
  - When a process terminates by calling exit() system call, its resources are deallocated by the OS. However, its entry in the process table must remain there until the parent calls wait(), because the process table contains the process's exit status.
  - A process that has terminated, but whose parent has not yet called wait(), then a process is remained as a zombie process since child status will use for parent's termination.
- Orphans process
  - Every process must have parent. If a parent process terminate before child process, the child process become orphan process.
  - The root process systemd (init in UNIX) will be its parent.

Process Termination
(Android Process Hierarchy)

- Because of resource constraints (limited memory), mobile operating systems may have to terminate existing processes based on a process's importance.
- The hierarchy of process classifications from most to least important is as follows:
  - Foreground process—The current process visible on the screen.
  - Visible process—A process that is not directly visible on the foreground but that is performing an activity
  - Service process—A process that is similar to a background process but is performing an activity that is apparent to the user (such as streaming music)
  - Background process—A process that may be performing an activity but is not apparent to the user.
  - Empty process—A process that holds no active components associated with any application

Process Termination
(Android Process Hierarchy)

- If system resources must be reclaimed, Android will first terminate empty processes, followed by background processes, and so forth.
- Processes are assigned an importance ranking, and Android attempts to assign a process as high a ranking as possible.