Cheng | Midterm 1: Chapter 3 | Study Guide

Chapter 3: Processes

Process Definition:

Process: A process is a program in execution. A process will need certain resources – such as CPU time, memory, files and I/O devices to accomplish its' tasks. These resources are typically allocated to the process while it is executing.

System: A system is a collection of processes; system code and user code being executed. These processes may execute concurrently. A system therefore consists of a collection of processes, some executing user code, others executing operating system code.

Thread: Multiple processes are called threads.

What is the difference between a process and a program?

A process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources. A program becomes a process when an executable file is loaded into memory. A program is static.

Process Layout in Memory:

Process is a program in execution. The status of the current activity of a process is represented by the value of the program counter and the contents of the processor's registers. The memory layout of a process is typically divided into multiple sections. The memory layout of a process is typically divided into multiple sections.

1. Text Section: The executable code. Fixed.
2. Data Section: Global variables. Fixed.
3. Heap Section: Memory that is dynamically allocated during program run time.
4. Stack Section: Temporary data storage when invoking functions (such as function parameters, return addresses, and local variables).

OS is responsible for making sure data and heap do not overlap as they dynamically change.

States & State Transitions:

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process.

1. New: The process is being created.
2. Running: Instructions are being executed.
3. Waiting: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
4. Ready: The process is waiting to be assigned to a processor.
5. Terminated: The process has finished execution.

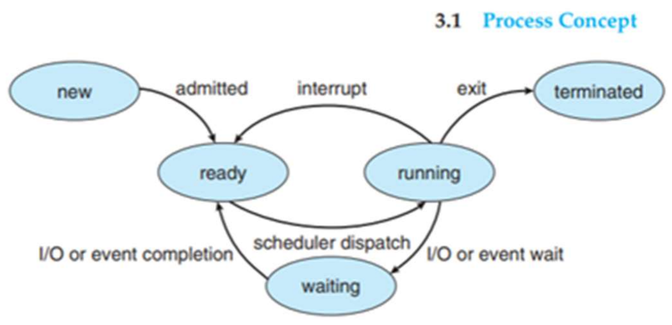

Figure 3.2 Diagram of process state.

Process Control Block:

Each process is represented in the operating system by a process control block (PCB) – also called a task control block.

Process State: The state may be new, ready, running, waiting, halted, and so on.

Program Counter: The counter indicates the address of the next instruction to be executed for these processes.

CPU Registers: The registers vary in number and type, depending on the computer architecture. This state information must be saved when an interrupt occurs.

CPU-Scheduling Information: This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory-management Information: May include value of base and limit registers, page tables, or segment tables, depend on memory system used by the OS.

Accounting Information: Amount of CPU and real time used, time limits, process numbers, …

I/O Status Information: List of I/O devices allocated to the process.

PCB serves as the repository for the data needed to start, or restarts, a process.

Types of Process:

Balancing the objectives of multiprogramming and time sharing also requires taking the general behavior of a process into account. In general, most processes can be described as either I/O bound or CPU bound.

CPU Bound : A process that speneds more of it’s time doing computations.

I/O Bound : A process that spends more of it’s time doing I/O than it spends doing computations.

Context Switching:

Interrupts or **System Call** cause the operating system to change a CPU core from its current task and to run a kernel routine.

When an interrupt occurs, the system needs to save the current context of the process running on the CPU core so that it can restore that context when its processing is done. The context is presented in the PCB of the process. We perform a **state save** of the current state of the CPU core, either kernel or user mode. And then **state restore** to resume operations.

Switching the CPU core performs a state save of the current process and state restore of a different process. This task is known as a **context switch**, it is pure overhead because the system does not do any useful work while switching.

Context Switching Speeds:

Switching speed depends on the complexity of the OS and the PCB, the more complex the longer it takes to switch.

Switching speed depends on the number of registers available in the hardware. Of course, if there are more active processes than there are register sets, the system resorts to copying register data to and from memory. Also, the more complex the operating system, the greater the amount of work that must be done during a context switch.

Long-term and Short Term Scheduling:

The operating system is responsible for managing the scheduling activites. The main memory cannot always accommodate all processes at runtime for multi-programmed OS. The operating system will need to decide on which process to execute next (short-term) and which processes will be brought to the main memory (long-term).

Schedulers:

Short-term Scheduler (CPU Scheduler) selects which process should be executed next and allocates CPU. This must be fast because it is invoked frequently.

Long-term Schedler (Job Scheduler) selects which processes should be brought into the ready queue. It might be slow because it is not invoked frequently. This LTS controls the degree of multiprogramming, which is basically when there are more processes than cores, and excess processes will have to wait until a core is free and can be rescheduled where the number of processes in memory is the degree of multiprogramming.

Process Creation & Termination:

- fork(): System call that creates a new process.
- exec(): System call used after fork() to replace the process' memory space with a new program.
- wait(): Returns status data from child to parent using wait(). The parent process may wait for terminaton of a child process by using the wait() system call. The call returns status information and the pid of the terminated process.

Inter-Process Communication:

Shared Memory:

An area of memory shared among the processes that wish to communicate. The communication is under the control of the users processes not the operating system. Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.

Message Passing:

Mechanism for processes to communicate and to synchronize their actions. Message system –processes communicate with each other without resorting to shared variables. IPC facility provides two operations: send(message) & receive(message). The message size is either fixed or variable.

A communcition link must be established! This can be either physcial, with a shared memory or logical with automatic or explicit buffering.