Processes

- Definition
- States
- Data structures (PCB)
- Switching and Scheduling
- Creation and Termination
- Monitoring

Process Definitions

- Definitions/terms
  - **Process** - a program "in execution". The smallest unit of work the OS explicitly keeps track of.
  - **Job** - the work submitted by the user to the OS. It can be broken down into one or more tasks which become processes when they become "in execution" state.
  - **Program** - code that a process executes.
    - Program is passive. Process is active.
    - Several processes can use the same program.

Process in Memory

![Process in Memory Diagram]

Process State

- As a process executes, it changes state
  - **new** - The process is being created
  - **running** - Instructions are being executed
  - **blocked/waiting** - The process is waiting for some event to occur. (It cannot run again until that event has occurred.)
  - **ready** - The process is ready to run (again) and is waiting to be assigned to a processor
  - **terminated** - The process has finished execution

Process States

![Process States Diagram]

Unix Process States

![Unix Process States Diagram]

Figure 3.15 UNIX Process State Transition Diagram
Process Context

- Context Switch
  - When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
  - Context of a process represented in the PCB
  - Context-switch time is overhead; the system does no useful work while switching
  - Time dependent on hardware support

Process Data Structures

- Process Control Block (PCB) - OS data structure to keep track of a process; one per process
  - Process Table - one for the whole OS, contains all the PCBs
  - Process ID (pid) – a unique identifier for a process

Process Control Block

- PCB Components (partial list)
  - Process state (ready, waiting, running)
  - Program counter (PC points to the next instruction to execute)
  - CPU registers
  - CPU scheduling information
  - Memory management information
  - Open files
  - I/O status information
  - Accounting information

Process Code Example

- Example:
  - User-submitted job: `cc myprog.c`
  - Resulting tasks: `cc myprog.c myprog.a as myprog.a myprog.o`
  - Resulting processes:
    - process 1 (PCB1 + code for C compiler + files)
    - process 2 (PCB2 + code for assembler + files)
    - process 3 (PCB3 + code for loader + files)
    - process 4 (PCB4 + code for profiler + files)

Switching between processes and OS kernel

From Process to OS

1. Involuntary: process is interrupted.
2. Voluntary: process calls the OS using one of the kernel system calls.

From OS to Process:

- OS scheduler selects next process to run and “context switches” to it. (use the PCB to setup the hardware to run that process)
- Special hw instruction to Load PC with value from PCB.
CPU Switch From Process to Process

Process Scheduling Queues
- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues

Ready Queue And Various I/O Device Queues

Representation of Process Scheduling

Schedulers
- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

Addition of Medium Term Scheduling

Intro to Processes
Schedulers (Cont)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow)
- The long-term scheduler controls the degree of multiprogramming

Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a tree of processes
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent’s resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

C Program Forking Separate Process

```c
int main() {
    pid_t pid;
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        perror("fork Failed");
        exit(2);
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait(NULL);
        printf("Child Complete");
        exit(0);
    }
}
```
A tree of processes on a typical Solaris

Process Termination
- Process executes last statement and asks the operating system to delete it (exit)
  - Process’ resources are deallocated by operating system
- Parent may terminate execution of children processes (abort)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates resulting in cascading termination

Interprocess Communication
- Processes within a system may be independent or cooperating
- Cooperating processes need interprocess communication (IPC)
- Two models of IPC
  - Shared memory
  - Message passing

We will not cover IPC in this course.

Intro to Processes