Processes

Reading:
Silberschatz
chapter 4

Additional Reading:
Stallings
chapter 3
Outline

- Process Concept
- Programming Types
- Process States
  - Process Creation
  - Process Termination
- Process State Diagram
- Process Control Block
- CPU Switching
- Queuing diagram
- Schedulers
- Context Switch
Process Concept

- **Definitions**
  - A process is a program in execution
  - An instance of a program running on a computer
  - The entity that can be assigned to and executed on a processor
  - A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system instructions

- **Key Points**
  - A program by itself is NOT a process
    - A program is a *passive entity* (file stored on disc)
    - A process is a *active entity* with associated resources and PC specifying the next instruction to execute
  - Two processes may be associated with the *same* program
    - Considered to be separate sequence; *e.g.* copies of same program
  - Processes are separated; no process can directly affect the state of another process
    - WWW browser, the shell program, compiled running program, *etc.*
Programming Types

- **Uniprogramming Vs multiprogramming**
  - Uniprogramming - Only one process at a time
  - Multiprogramming
    1. Multiple process at a time
    2. Which process gets physical resources of machine?
      - Preemptive multitasking
      - Fairness – all process must get fair share of the CPU
Processes

- **Execution model**
  - OS components → Organized into number of sequential processes
  - Each process → Block of code with a pointer showing next instruction to be executed
  - How can several processes run on one CPU?
  - OS makes this happen by ensuring
    - Fair scheduling → each process gets fair chance to run
    - Protection → processes do not modify each others state
Process State

As process executes it changes its state

- **New**
  - The process is being created, resource acquisition

- **Ready**
  - The processes that are prepared to execute at next opportunity

- **Running**
  - The process that is currently being executed by CPU

- **Waiting**
  - The process is waiting for some event to occur

- **Terminated**
  - The process has completed execution
Above state names are arbitrary and OS dependent
The state they represents is found in all systems
Only ONE process can be *running* but several process may be *ready* and *waiting*
Process Control Block (PCB)

Each process is represented in OS by PCB. The PCB contains pieces of information associated with each process, including:

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information
Process Control Block (PCB)

- Pointer
- Process state
- Process number
- Program counter
- Registers
- Memory limits
- List of open files
CPU Switching

- process $P_0$
- operating system
- process $P_1$

- executing
- idle
- executing
- interrupt or system call
- save state into PCB$_0$
- ... 
- reload state from PCB$_1$
- interrupt or system call
- save state into PCB$_1$
- ... 
- reload state from PCB$_0$
- idle
Process Scheduling Queues

- **Job queue**
  - An instance 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

- **Process migration between the various queues**
Ready And Various I/O Device Queues
Queuing Diagram

ready queue -> CPU

I/O -> I/O queue -> I/O request

time slice expired

child executes

fork a child

interrupt occurs

wait for an interrupt
Multiple Blocked Queues

(b) Multiple blocked queues
Schedulers

- **Long-term (job) scheduler**
  - Selects a process from pool
  - Loads them into memory for execution
  - Controls the degree of multiprogramming
    - # process in memory, stable – invoke during departure
  - Can afford to take more time in decision – long execution time
  - I/O bound process, CPU bound process, best combination

- **Short-term (CPU) scheduler**
  - Selects from the processes that are ready to execute
  - Allocates *CPU to one of them*

  - Some time-sharing OS **no** long-term scheduler, simply put new process in memory for short-term scheduler
    - *e.g.* UNIX, MS Windows
Medium Term Scheduling

- Additional intermediate level of scheduling
  
  temporarily remove processes from memory, active contention of CPU
Context (Process) Switch

- CPU switches to another process
  - Save the current context/state of the old process
  - Load/restore the saved context/state for new process
  - The context is represented in PCB of process

- Context-switch time is overhead; the system does no useful work while switching

- Time dependent on hardware support
  - *Sun UltraSPARC* provides multiple set of registers
  - Context switch here simply requires changing the pointer to current register set (if processes > registers, save to memory)

- OS should *masks/disables all interrupts* while saving the process state, Implementation?
CPU Control

- Most Computers has ONE CPU
- When a process is running, scheduler/dispatcher cannot run, OS May loose control
- How does the OS regain the control of CPU?
Process Memory Components
Process Creation

- **Parent process → children processes → other processes; tree of processes**

- **Resource sharing**
  - Parent and children share all resources
  - Children share subset of parent’s resources
  - Parent and child share no resources

- **Execution**
  - Parent and children execute concurrently
  - Parent waits until children terminate

- **Process Identifier**
  - Most OS (UNIX, Windows), process → unique pid
  - Unique Integer
Process Creation

Process Tree: Typical example from Solaris system
Process Creation

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it

- UNIX examples
  - `fork` system call creates new process
  - `exec` system call used after a `fork` to replace the process’ memory space with a new program
Process Creation

#include <unistd.h>  /* Symbolic Constants */
#include <sys/types.h>   /* Primitive System Data Types */
#include <stdio.h>      /* Input/Output */

int main()
{
    pid_t pid;  /* variable to store the child's pid */

    /* create a new process */
    pid = fork();

    if (pid < 0) {/* error occurred, fork returns -1 on failure */
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }

    else if (pid == 0) {/* child process, fork() returns 0 to the 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);
    }

    return 0;
}
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    // allocate memory
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    // create child process
    if (CreateProcess(NULL, // use command line
                        "C:\WINDOWS\system32\mspaint.exe", // command line
                        NULL, // don't inherit process handle
                        NULL, // don't inherit thread handle
                        FALSE, // disable handle inheritance
                        0, // no creation flags
                        NULL, // use parent's environment block
                        NULL, // use parent's existing directory
                        &si,
                        &pi))
    {
        printf(stderr, "Create Process Failed");
        return -1;
    }

    // parent will wait for the child to complete
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    // close handles
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
Process Termination

- Process terminates when it executes last statement and asks the OS to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - 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
  - Parent is exiting
    - Operating system does not allow child to continue if its parent terminates
    - Cascading termination