Processes

COMP755
Advanced Operating Systems

OS Structure

- An OS has many parts.
- The Kernel is the core of the OS. It controls the execution of the system.
- Many OS features run outside of the kernel, such as the shell.

OS Structure

Modes of Execution

- Kernel Mode
  - more privileged mode
  - has complete control of the processor and all its instructions, registers, and memory
  - not desirable for user programs
- User Mode
  - less privileged mode
  - user program typically execute in this mode
  - parts of the OS run in this mode

Typical Functions of an OS Kernel

- Process Management
- Memory Management
- I/O Management
- Support Functions

Non-Kernel OS

- Shell – user interface to the OS. In Microsoft Windows, this is called the Explorer (not the Internet Explorer).
- File Sharing
- Remote Procedure Calls
- Utilities
The Process

- All multiprogramming OSs are built around the concept of processes.
- A process is something akin to a program.
- Some programs have multiple processes and the OS has several processes.
- A process has:
  - memory
  - ability to use the CPU (a thread)
  - resources allocated to it

Multiple Processes

- There are usually many processes existing in the system.
- If there is only one CPU, then only one process can be running at any one time.
- Other processes might be waiting to run.
- Some (most) processes will be blocked waiting for something to happen (i.e. I/O)

Sharing the System

- Most programs perform a lot of I/O and therefore spend a lot of time waiting for I/O to complete.
- When a program is waiting for I/O, another program can be running.
- Some programs wait for a very long time for input (such as input from a keyboard, mouse or network)

“From a programmer's point of view, the user is a peripheral that types when you issue a read request.”

- P. Williams

Process Creation

- When a process is created, the OS
  - Builds the data structures that are used to manage the process
  - Allocates address space in main memory to the process
- When does a process get created?
  - When a program is started
  - Created by OS to provide a service to a user (ex: printing a file)
  - Spawned by an existing process

Process Termination

- User quits an application (e.g. word processor)
- Task completes
- Error and fault conditions
A Five-State Process Model

Queue View of Process States

**Process States**
- **Running**: The process is being executed
- **Ready**: The process is prepared to execute when given the opportunity
- **Blocked**: The process can not execute until some event occurs, such as the completion of an I/O operation

**Process Transitions**
- **New** → **Ready**: When a process becomes ready to be executed.
- **Ready** → **Running**: When the CPU is available, the dispatcher selects a new process to run
- **Running** → **Ready**: The running process has run long enough for now; the running process gets interrupted because a higher priority process is in the ready state (preempted)
Queuing Model Multiple Blocked Queues

Dispatcher
- A part of the OS that gives the processor from one process to another
- Selects a process from the queue to execute after interrupt or process termination
- Prevents a single process from monopolizing the processor time

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
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Operating System Control Structures
- An OS maintains several tables for managing processes and resources:
  - Memory tables
  - I/O tables
  - File tables
  - Process tables

Process Control Structures
- Process image: the collection of program, data and attributes (process control block)
- Process control block: the collection of attributes of the process that are used by OS for process control

Process Control Block (PCB)
- Process Identification Information
  - Processor Identifier (PID)
    - 16 bit integer unique for each process
  - PID of the parent process
  - User identifier
    - who is responsible for the job
PCB Contents

- CPU State Information
  - What the CPU needs to run this process.
  - User-visible registers
  - Program Counter
  - PSW
  - Stack pointers

- Scheduling and state information
  - Process state
  - Priority
  - Event for which the process is waiting (if blocked)

- Process relationships
  - May hold pointers to other PCBs for process queues, parent-child relationships and other structures

PCB Contents

- Inter-process communication (IPC)
  - May hold flags and signals for IPC

- Process privileges

- Memory management
  - Pointers to segment/page tables assigned to this process

- Resource ownership and utilization
  - Resources in use: opened files, I/O devices

Process Creation

- Assign a unique Processor ID (PID).
- Allocate RAM for the PCB
- Initialize the PCB
- Allocate RAM for the program (code, data, stack, and heap)
- Copy the program into the RAM
- Put the PCB on the list of PCBs

Creating a new Process

- In C or C++, you can create a new process with the fork function
  - int fork();
- When fork is called, the RAM of the program is copied to another location is RAM. A new process is created to run in the new program address space. The new process is put on the ready list.

fork function

- The return value of the fork function is:
  - -1 = Error
  - 0 = This is the process that was just created.
  - Number = This process is the parent that just executed the fork function. The number returned is the PID of the child process.
Fork Action

Parent Program:
- `p = fork();`
- `code`
  - `data`
  - `p = 567`
  - `x = 3`
  - `y = 5`
- `stack`
- `heap`

Child Program:
- `p = fork();`
- `code`
  - `data`
  - `p = 0`
  - `x = 3`
  - `y = 5`
- `stack`
- `heap`

/* Example fork program */

```c
int counter = 0; /* loop counter */
int pid; /* processor ID of child */
int waitinc; /* wait time in seconds */
char id; /* display identifier */
pid = fork();
if (pid < 0) { cout << "Fork error"; exit(8); }
else if (pid == 0) { // if this is the child process
  id = 'C';
  waitinc = 1;
} else { // this is the parent process
  id = 'P';
  waitinc = 2;
}
for (i = 0; i < 8; i++) {
  counter++;
  cout << id << " " << counter << endl;
  sleep(waitinc);
}
cout << "end of " << id << endl;
```

Sharing Memory

- Creating a new process makes a completely new address space with a copy of the original program.
- None of the variables in the parent copy are accessible by the child process.
- Programmers can make a shared memory segment to communicate.

**shmget**

- The `shmget` function creates a shared memory segment.
  ```c
  int shmget(int key, int size, int flags);
  ```
  - `key` is the id of the existing shared memory segment or `IPC_PRIVATE`.
  - `size` is the size of the segment to make.
  - `flags` should be `IPC_CREAT|0666`
  - Returns the shared memory id

**shmat**

- The `shmat` function gets the address of a shared memory segment.
  ```c
  shmat(shared_id, 0, 0);
  ```
  - `shared_id` is the return value from `shmget`.
  - Returns the address of the shared memory segment.
**shmctl**

- Shared memory control function can be used to delete a shared memory segment.
  
  ```c
  shmctl(shared_id, IPC_RMID, NULL);
  ```

- `shared_id` is the return value from `shmget`.
- `IPC_RMID` is the function code to delete a shared segment.

---

**Programming Note**

- Shared memory segments are permanent data structures. They do NOT go away when the program terminates.
- Be sure to delete any and all shared memory segments you create.

---

**Multiple Process Conflicts**

- A process or thread can be halted at any time by the OS to run another process or thread.

---

**X++ in Machine Language**

```c
LOAD  R1,X
ADD   R1,1
OS switches to another process
LOAD  R1,X
ADD   R1,1
STORE R1,X
OS switches to another process
STORE R1,X
```