CS340 – Exam 1

**Lecture 5:**

**Process Control Block (PCB):** The PCB of a process provides all *key information* about the process to the OS. The access to the PCB of a process is done using a unique Process ID (PID).

**Key Information:**

1. Process States
2. Program Counter
3. CPU Scheduling Information (priority, pointers to scheduling queue).
4. Memory-Management Information (base and limit register, page table, segments tables)
5. Accounting Information
6. I/O Status Information (list of I/O devices allocated, list of open files)
7. CPU Registers (AC, PC, general purpose registers) content

**Lecture 6:**

**Mode Switch:**

1. Saves the context of the processor.
2. Interrupt Handler Routine is called. The IHR checks the cause of the interrupt and may or may not resolve it. The main task of the IHR is to protect the PCB of the process.
3. The processor switches from user mode (1) to system mode (0).

Example:

**Full Context Switch:**

1. Saves the context of the process.
2. Update the PCB of the process that is running currently.
3. Move the PCB to the appropriate queue.
4. Selects another process for execution.
5. Update the PCB of this process.
6. Update the memory management data structures.
7. Restore the context of the process.

Example: wait() & exit().

**Process States:**

1. **New** – Process that has just been created that is not yet admitted into pool of executable processes.
2. **Ready Swap** – Process is the Secondary Memory and is ready for execution once loaded to MM.
3. **Ready (active)** – Process in MM that is ready for execution.
4. **Running** – Process that is currently being executed.
5. **Blocked** – Process in MM that is awaiting an event. Not enough resources.
6. **Blocked Swapped** – The process is in SM and may wait for an event to occur.
7. **Terminated** – Process finished execution.

**State Transitions & State Diagram:** 🡪 (Points to) 🡨(is pointed at/points back)

New: 🡪 Ready

Ready Swapped: 🡪 Ready and Ready 🡨

Ready: 🡪 Run and Run 🡨. Also, 🡪 Ready Swapped and Ready Swapped 🡨. Blocked 🡪 to it.

Run: 🡪 Blocked. 🡪 Ready and Ready 🡨. 🡪 Terminated.

Blocked: 🡪 to Ready. 🡪 Blocked Swapped and Block Swapped 🡨.

Blocked Swapped: 🡪 Ready Swapped.

Terminated: 🡨 Run.

Main Memory 🡪 Secondary Memory: Free up space in MM.

SM 🡪 MM: Increased level of multiprogramming

MM 🡨🡪 SM: Contiguous Allocation (Medium Term Scheduler)

Ready 🡨🡪 Run: Short term scheduler.

**Lecture 7:**

**Process Operations:**

**Activate –** Restart the process from the point it was suspended. Moved from Swapped-Out in SM to active state.

**Swapped –** Control the number of processes in MM, the degree of multiprogramming. A process remains suspended until another process activates it. High-priority operation.

**Release –** Blocked to Ready.

**Process Creation:**

**In terms of execution:**

1. The parent continues to execute concurrently with its children.
2. The parent waits until some or all its children terminated.

**In terms of resource utilization:**

1. Child may obtain resources from OS.
2. Subset of resources of parent (good for deadlock handling).

**In terms of the address space:**

1. The child is a duplicate of the parent (same address space & variables with the same values).
2. The child is a separate program (different address space)

**System Calls used to create a process:**

1. Fork() – Creates process.

Returns -1: Cannot create process

Returns 0: Returned value to the child.

Returns >0: The PID of the child

1. Exce(…) – Allows a process to overwrite its address space.
2. Wait() – Allows the parent to detect a child’s termination.

Init – Manages all user activity and create the initial user process for each user.

**Process Termination:**

When process is terminated, it asks the OS to delete it by exit (). The process may return data to its parent through wait (provides PID). A process can cause the termination of another process using abort – usually only done by parent.

A parent can terminate one of its children if:

1. The child exceeded the use of resources.
2. The task of the child is no longer needed.
3. The parent is exiting; the OS doesn’t allow child to continue if parent is terminated.

If a parent terminates, the child is assigned the init process as new parent.

Echo $? – Displays the exit code of the last process.

Zombie State – The state of a process between the time it has exited and the time when the parent has gotten its exit code by calling wait ().

Cascading termination – VMS – OS terminates all children when the parent is terminating.

**Lecture 8:**

1. Processes unaware from each other: Not intended to work together. OS needs to be concerned about competition for resources.

Example: Multiprogramming of multiple independent processes.

Potential Problems: ME, Deadlock, Starvation.

1. Processes indirectly aware of each other: Not necessarily aware of each other but share access to some object, such as I/O Buffer.

Problems: ME, Deadlock, Starvation, Data Coherence (Reader/Writer problem).

1. Processes directly aware of each other: Can communicate with each other by name.

Problems: Deadlock, Starvation.

**Critical Section**: is a segment of code in which the process may be access common variables, shared data, etc.

**Mutual Exclusion Condition:** One process allowed in the CS at a given time.

**Critical Section Problem:** design a protocol that processes can use to cooperate and properly use the data of the critical section.

**Constraints on acceptable solutions to CS problem:**

**Mutual Exclusion:** with respect to CS. At a given time, only one process can be in the CS.

**Progress Condition:** if no process is in the CS and there are processes trying to enter their CS, then only processes competing for the CS should participate in the decision of which will enter the CS next and the decision must be made in finite time.

**No Starvation:** no process should be postponed for an indefinite period.

**Lecture 10:**

**Test and Set (TS):**

1. Atomic – Without interrupt
2. Pros – Works for any # of P
3. Pro – Works in multiprogramming and multiprocessing
4. Con – Starvation.

Implementation of ME:

Lock = F;

P2:

While(true){

While(TS(lock)){}; - Trap

CS;

Lock = F;

RS

}

ME: Assume Pi is in CS;

If Pj attempts to enter, it will BW

For Pj to exit BW

TS(Lock) must be F 🡪TS=F 🡪 Lock=F;

Lock = F; is done by Pi after it exits the CS.