**CS 340 Midterm Review Sheet**

**Interrupting a Process :**

Interrupt - the occurrence of an event signaled by an interrupt from the hardware or software. The signal is sent to the processor that temporarily stops a running program and allows a special program, an interrupt handler to run instead.

1) Asynchronous – a signal from I/O module (hardware).

2) Synchronous – the direct result of execution of the current instruction being executed.

3) System Call – controlled of execution from unprivileged code to the OS via software.

**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 (Ready, Active, Wait)
2. Program Counter also called an instruction pointer and known as a status register. It contains the address of the instruction being executed at the current time. As each instruction gets fetched, the program counter increments by one. Also contains the address of the next instruction to be fetched. It indicates where a computer is in its program sequence.
3. CPU Scheduling information (priority, pointers to scheduling queue). It’s the process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resources like I/O, thereby making full use of CPU. It is what carries out the scheduling activity and when the CPU becomes idle, it’s the job of the CPU scheduler to select another process from the ready queue to run next, keeping it busy. It handles the removal of the running process from the CPU and selection of another process.
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.

Extra:

1. Name of the processes
2. Resources allocated to the process
3. Memory which is provided to the process
4. Process ID or a Identification Number which is given by the CPU when a Process Request for a service

**Mode Switch-** This is what is referred to when the CPU changes privileged levels. The kernel works at a higher privilege than a standard user task. In order for the user task to access things controlled by the kernel, it’s necessary for a mode switch occur. These instructions are system calls (OS hides privileged instructions as system calls).

1. Saves context of the processer

2. Sets the PC to the starting address of an OS, the interrupt handler routine

3. Switches from user mode to system mode

Examples: I/O operations

**Full Context Switch-** Process of storing and restoring state of a process or thread so that execution can be resumed the same point at a later time. This enables multiple processes to share a single CPU and is an essential feature of a multitasking operating system. It is also known as the switching of the CPU from one process or thread to another. This causes the contents of the CPU registers and instruction pointer to be saved. The register and instruction pointer for the new task will then be loaded in to the processor and execution of the new process will start/resume. The old program is no longer executing but it’s state is saved in memory for when the kernel decides that it is ready to execute it again. This is what gives the illusion of multitasking.

1. Saves the context of the process

2. Update the PCB of the process that is currently running

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

Examples: wait(), exit()

**Process States:**

1. New – a process that has just been created that is not yet admitted into pool of executable processes.
2. Ready Swap – the process is in secondary memory, but is available for execution as soon as it is loaded into main memory.
3. Ready (active) – the process is in main memory and available for execution .
4. Running – the process is currently being executed.
5. Blocked – the process is in main memory and awaiting an event. The process doesn’t have all the resources it needs (for ex: I/O Op waiting for an event)
6. Blocked swapped – the process is in secondary memory and it may wait for an event
7. Terminated – the process has finished execution

New 🡪 Ready: admits process in MM

Ready 🡪 Run: schedules process

Run 🡪 Ready: scheduling algorithm preempted – preparing for next process

Run 🡪 Blocked: waits for event, I/O privileged instruction

Run 🡪 Terminate: Normal (exit() syscall to OS) vs. Abnormal (by OS, user, parent)

Ready 🡪 Ready Swapped: suspends to secondary memory (frees up space in main memory)

Ready Swapped 🡪 Ready: activates to main memory (increased level of multiprogramming

\*medium term scheduler – MM to SM

**Normal termination** - when a process terminates normally when its program signals it is done by calling exit.

**Abnormal termination** - A termination to a process brought about by the operating system when the process reaches a point from which it cannot continue like when the process attempts to obey an undefined instruction.

**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.