CS340 Midterm Review

Operating Systems-

* Interface between the hardware and programs, and programs and users
* OS doesn’t directly execute a process/ program

Batch Processing System-

* Jobs with similar needs are batched together and run as a group (Sequential Execution)
* Major task of OS was to automatically transfer control from one process to the next
* Batch systems are characterized by lack of interaction between user and job during execution

Batch File- Text file containing a series of commands for the OS, executed without interaction.

Multiprogramming/ Multitasking Systems-

* One CPU with multiple processes ready for execution- partial or entire process in main memory
* Competition among processes for resources (memory, CPU, etc)

Multiprocessing Systems-

* Several processors are used on a single computer system to increase processing power of the machine
* Processes share computer bus, clock, and sometimes memory

*Symmetric Model:* All CPU’s are equal

*Asymmetric Model:* CPU’s have master/slave relationship

Timesharing System-

* Special multiprogramming system, supports multiple terminals- one for each active user
* Highly Interactive with FCFS scheduling algorithm

Real-Time Systems- used for rigid time requirements (eg. Lynx OS and RT Linux)

* Hard real-time: guarantees critical task is done in time. Each process has a constraint for starting/stopping (eg. army, airplane controls, robotics)
* Soft real-time: Constraint on one process at a time (eg. multi-media)
* Real-time systems use more ROM, less RAM. Minimal interrupts and interaction with user
* No concept of virtual memory

Starting a Computer-

When the CPU is started, it branches to a fixed memory location, this hardwired address points to the ROM which contains the **Bootstrap Program:** Initial program needed for the computer to start running. This program initializes all aspects of the system- locating the OS kernel and loading it into memory. Then the OS starts executing the first process and waits for some event to occur.

Interrupts- Signals the occurrence of an event.

* From the hardware: by a signal sent to the CPU.
* From the software: by executing a system call.

*Synchronous:* direct result of the current instruction executed by the CPU

eg. Over/ under flow, illegal reference, page fault

*Asynchronous:* From the hardware/scheduler eg. Mouse, keyboard

System Calls- Interrupt from a process; interface between user **process** and OS.

Dual-Mode Operation-

* Implemented by the hardware for protection
* A mode bit is added to the hardware of the computer to indicate the current mode

*User-Mode:* bit is set to 1 for an operation done on behalf of the user

*System/ Monitor Mode:* bit is set to 0 for an operation done on behalf of the OS

Privileged Instructions- Can only be executed in monitor mode (eg. I/O Operation)

Operating System Components- The OS is partitioned into system components with specific tasks

Command Interpreter System/ Program- interface between the user and OS

* Can be included in the kernel or above the kernel
* Commands can come from files or directly from the terminal
* The command interpreter contains executable code either outside or part of the CI, which can be implemented through special programs.

Single-tasking System (MS-DOS)-

* The CI is invoked when the computer is started
* MS-DOS loads the program to be run into the main memory, overwriting part of the CI
* PC is set to the first instruction of the loaded program
* Program runs- either an error causes a trap, or the program executes a system call to terminate
* CI resumes execution
* The residual piece of the CI reloads the rest of the CI from the disk.

TSR (Terminate & Stay Resident Call)-

* Computer system call in DOS, returns control to the system as if the program has quit but keeps the program in memory
* Can be reactivated by hardware/ software interrupt
* Partially overcame MS-DOS’s limitation of only executing one program at a time

CI- Multi-tasking (UNIX)-

* The CI in UNIX is a process that runs in user mode
* Each line of the shell is parsed to obtain strings of the command and the parameters
* The shell of the user’s choice (CI) is run when a user logs on to the system
* The shell either waits for the process to finish or runs the process in the background

Modern OS Architecture-

* *Layered Structure:* Given the hardware support, the OS can be broken into smaller pieces, creating a **modular OS**- build levels of OS on top of lower layers

Lowest layer= the hardware, highest layer= the user interface

* This layered system replaces the old vertical architecture with a horizontal one. Makes it easier to debug program because you can locate where the problem is.

Micro-kernel-

* Introduced in Windows NT architecture
* Represents the most used and fundamental part of the OS, functions as a message exchange

HAL (Hardware Abstraction Layer)-

* Isolates the OS from platform specific hardware differences
* Provides the support for Symmetric Multiprocessing

Win32-

* User mode processes that enables Windows to run programs developed for other OS’s
* The most important subsystem
* The Win32 API represents the native environment for Windows NT and 2000

Windows NT- introduced the micro kernel concept

**Section 2:**

PCB (Process Control Block) -

* Provides all the key information of a process to the OS using a unique PID
* Interrupt Handler Routine

Interrupting a Process-

* Asynchronous Interrupt
* Synchronous Interrupt
* System Call

Mode Switch-

* Saves the context of the current program being executed
* Sets the PC to the starting address of an OS program, the **interrupt handler:** protects the PCB of a process. Usually the only process that can modify information inside the PCB
* Meanwhile, the processor switched from user to system mode

Full Context Switch-

* Saves the contents of the processor
* Updates the PCB of the process that is currently in the Running state
* Moves the PCB of the process to the appropriate queue
* Selects another process for execution
* Updates the PCB of the new process and memory management data structures
* Restores the context of the processor to that which existed at the time the selected process was last switched out of the Running state

Process States-

* **New:** A process that has just been created but not yet admitted to the pool of executable processes
* **Ready Swapped:** The process is in secondary memory but available for execution as soon as it’s loaded into the main memory
* **Ready (active):** The process is in the main memory and available for execution
* **Running:** The process is currently being executed.
* **Blocked:**  The process is in the main memory and waiting for an event, doesn’t yet have all its resources.
* **Blocked Swapped:** The process is in secondary memory and it may wait for an event.
* **Terminated:**  The process has finished execution

Process Transitions-

New -> Ready: Admit process in main memory

Ready -> Run: Schedule process

Run -> Ready: (Scheduling Algorithm) Preempted- preparing for next process

Run -> Blocked: Wait for event, I/O OP (privileged instruction)

Run -> Terminate: Normal (exit() system call to OS) vs Abnormal

Ready -> Ready Swapped Out: Suspend

\*\*\*See Diagram\*\*\*

**Process Operations-**

*Process Creation:* A process can create new processes called **children** of that process

* PCB is created and updated
* Process gets a unique ID
* Structures are created for process (eg. tables)
* Execution Stack
* Queues updated
* Address space is allocated for instruction and data

*Process Termination:* When a process finishes executing it asks the OS to delete it by using the exit system call

Atomic Operation- An operation without interrupts and context switching (eg. write&read memory cells)

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 of time.

**Two Process-Software Solution-**

No support of the hardware, operating system or programming language level is assumed.

* **Attempt 1:**Processes communicate using common variable turn. If turn = i then Pi is allowed to execute.

*The Progress Condition is not satisfied.*

* **Attempt 2:**Each process should have its own key to the CS so that if one process is executing outside the CS the other one is able to get in to its CS.

*Mutual Exclusion and No Starvation are not satisfied*

* **Attempt 3:** First the process sets its flag to true to signal that is ready to enter the CS. Next, the process checks if the other process is not also ready to get in the CS.

*No deadlock is not satisfied.*

* **Peterson Solution:**

The state of both processes is provided by the global array variable 'flag'.

Also we need to impose some order on the activities of the two processes. The variable 'turn' solves the simultaneity conflicts.

*This is the correct solution*

**Test and Set (TS) - Takes and returns a Boolean value**

Atomic instruction, works in any environment with any number of processes (Only works for two processes in Peterson). TS of a memory location causes the content of the specified memory location to be loaded into the CPU register and the memory to be written with a value of True.

**Definition of the Test-and-Set instruction:**

boolean TS (target:boolean) {

TS = target;

target = true;

}

* **Mutual-Exclusion implementation with TS:** Satisfied
* Assume P[i] is in the CS -> lock = T
* If P[j] attempts to enter the CS, P[j] will BW
* For P[j] to exit BW, TS(lock) = F -> TS = F -> lock = F
* The only way for lock = F is through P[i] after P[i] exits the CS
* **No Starvation:** Not Satisfied
* **No Delay:** Satisfied

Questions:

1. UNIX Environment

**(A) What does the BIN directory contain?**

The BIN (binary) directory contains executable files and most UNIX commands

**What does the DEV directory contain?**

The DEV directory contains special device files for all devices.

**(B) Discuss the creation of a process from the point of view of ADDRESS SPACE.**

When a process is created from a parent using the fork() command, the child is given its own address space, however the child possesses the same address space information as the parent address space, but gets executed independently from each other.

**(C) What is the name of the system call (command) that can overwrite the default address?** exec()

**(D) What happens to a child process if its parent is terminated?**

If the parent is terminated or killed, the child process will be adopted by the init process.

**(E) What are the return values for the FORK() command, and what are their meanings?**

Fork returns a -1 if the process cannot be created.

Fork returns a 0 to the child process, meaning that the child process has been created.

Fork returns a 1 to the parent process along with the child process ID.

**(F) Define a ZOMBIE process.**

The state of a process between the time it has exited and the time when its parent received the exit code by calling the wait command.

**(G) Explain the main difference between a MODE SWITCH and a FULL CONTEXT SWITCH. Give examples of each type of switch.**

The main difference is that Full Context Switch switches to a new process after an interrupt, and Mode Switch is an interrupt in the current process.

Mode Switch (Minimal Delay): During mode switch the process context is saved. The mode changes from user to system. The interrupt is handled. The mode is changed from system to user mode. The process continues execution. Eg. I/O Operation

Full context Switch (Longer Delay): During Full Context Switch the process context is saved. The mode changes from user mode to system mode. The process is placed in the appropriate queue. A new process is then scheduled for execution. The system changes from system mode to user mode. Full context switch occurs due to scheduling and creates a much bigger delay since the first process must wait its turn to complete executing. Eg. exit() wait()

**Process Control Block**

MARICSS – Memory Management, Accounting Information, CPU Registers, I/O Status Information, and Program Counter, Process State, Scheduling Information.

**What information is stored in the PCB?**

Process Identifiers: Parent Process Id, Process Id, User Id.

Processor State Information: Process State, Program Counter, Status Information

Process Control Information: Process State, Event, Memory Management, Privileges, Priority, Resources, Scheduling Information.

Mid-term 2

1. UNIX Environment

**(A) What does the BIN directory contain?**

The BIN (binary) directory contains executable files and most UNIX commands

**What does the DEV directory contain?**

The DEV directory contains special device files for all devices.

**(B) Discuss the creation of a process from the point of view of ADDRESS SPACE.**

When a process is created from a parent using the fork() command, the child is given its own address space, however the child possesses the same address space information as the parent address space, but gets executed independently from each other.

**(C) What is the name of the system call (command) that can overwrite the default address?** exec()

**(D) What happens to a child process if its parent is terminated?**

If the parent is terminated or killed, the child process will be adopted by the init process.

**(E) What are the return values for the FORK() command, and what are their meanings?**

Fork returns a -1 if the process cannot be created.

Fork returns a 0 to the child process, meaning that the child process has been created.

Fork returns a 1 to the parent process along with the child process ID.

**(F) Define a ZOMBIE process.**

The state of a process between the time it has exited and the time when its parent received the exit code by calling the wait command.

**II (G) Explain the main difference between a MODE SWITCH and a FULL CONTEXT SWITCH. Give examples of each type of switch.**

The main difference is that Full Context Switch switches to a new process after an interrupt, and Mode Switch is an interrupt in the current process.

Mode Switch (Minimal Delay): During mode switch the process context is saved. The mode changes from user to system. The interrupt is handled. The mode is changed from system to user mode. The process continues execution. Eg. I/O Operation

Full context Switch (Longer Delay): During Full Context Switch the process context is saved. The mode changes from user mode to system mode. The process is placed in the appropriate queue. A new process is then scheduled for execution. The system changes from system mode to user mode. Full context switch occurs due to scheduling and creates a much bigger delay since the first process must wait its turn to complete executing. Eg. exit() wait()

**Process Control Block**

MARICSS – Memory Management, Accounting Information, CPU Registers, I/O Status Information, and Program Counter, Process State, Scheduling Information.

**What information is stored in the PCB?**

Process Identifiers: Parent Process Id, Process Id, User Id.

Processor State Information: Process State, Program Counter, Status Information

Process Control Information: Process State, Event, Memory Management, Privileges, Priority, Resources, Scheduling Information.

**(H) Test and Set**

p[i] (Lock is initially set to false)

while(true){

while(TS(LOCK)){}

CS

LOCK=FALSE;

}

Assume Process 1 is in the CS. For Process 1 to be in the CS, Lock must be set to true.

If Process 2 tries to enter the CS, Process 2 will busy wait. For Process 2 to exit BW, TS (LOCK) must return false.

For TS (Lock) to return false, Process 1 must finish the CS, which will set TS (LOCK) to false.

Since this is an atomic implementation, no Process can get into the CS while another process is checking the LOCK. The Process cannot be interrupted.

**What if it was non-atomic?**

If it was non-atomic, the process can be interrupted during the CS, however, Lock will remain the same since the lock is only changed after the CS, which does not allow another process to enter CS.

**Q: Children wait to get their gift.** Each child process gets its turn[i] by computing number[i]. A clown has in his hat --- each ball has a different number, from 1 to 10. After all children have their turn set, the clown is --- are the gifts (there are 10 gifts). The clown picks a random ball, and gives the gift to the child whose number is the same with the number of the ball (after that, he throws away the selected ball).

Variables:

turn[] = 0; number[] = 0; called[] = 0; ballsNumber = 10; i = 1,.......10 (N=10)

child(int i){

number[i] = 1+max(number[1],number[2],....number[N});

turn[i] = number[i];

while(!called[i]) {busyWait;}

getTheGift(); //sleep

ballsNumber--;

goHome(); //sleep

}

clown(){

while(ballsNumber>0){

pickABall(); //sleep

for(int j=1; j<=N; j++){

if(turn[i]==numberOnBall){

called[j]=true;

giveTheGift(); //sleep

} } }

leave;

}//clown

\*ALL CHILD PROCESSES EXECUTE CONCURRENTLY.

**(A) Is it possible for 2 children to compute the same value for number[i]? Explain why and give the execution sequence that can show it.**

Yes it is possible for 2 children to have the same value for number[i]. This is possible because this is a high level language. The computer must first load the number[i], and the number[i] must be calculated then stored. However, another thread can load number[i] before the first thread stores it. There is possibility for data coherence.

P1=loads number[i] = 0

p1=calculates number[i] = 1

p2=loads number[i] = 0 (since p1 did not store number[i], p2 loads what is already in memory)

p2=calculates number[i] = 1

p1=stores number[i] = 1 p2=stores number[i] = 1

**(B) Consider that 6 children already computed their number[i]. Give an execution sequence by which after these 6 children computed their number, the largest computed number[i] = 4.**

P1=loads number[i] = 0

p1=calculates number[i] = 1

p2=loads number[i] = 0 (since p1 did not store number[i], p2 loads what is already in memory)

p2=calculates number[i] = 1

p1=stores number[i] = 1

p2=stores number[i] = 1

p3=loads number[i] = 1

p3=calculates number[i] = 2

p4=loads number[i] = 1

p4=calculates number[i] = 2

p3=stores number[i]=2

p4=stores number [i] =2

p5=loads number[i] =2

p5=calculates number[i] = 3

p5=stores number[i]= 3

p6=loads number[i] =3

p6=calculates number[i] = 4

p6=stores number[i]= 4

**(C) On the hypothesis that each child has a different number[i], is it possible for children to compete for the same give (because turn[i] values are the same)? Explain. If yes, give the execution sequence.**

No, since each child has a different number, the turns will be a different number as well. Since number is different for each child, and turn[i]=number[i], there is no way for the turns to be the same. It is however possible for the turns to be the same, if the number[i] is the same.

**(D) Consider that at this point, all children have their turn[i] updated (are done with the execution of turn[i]=number[i]) Is it possible for a child to starve (never be called), by busy waiting in the while loop? Explain. If yes, give the execution sequence that will show it.**

No, it is not possible for this to happen. If the children have the same turn[i], they will be called, they will simply compete for the same cookie. If the children all have different number[i], they will all have different number[i] as well, so each one will be called.

**(E) Under the hypothesis that each child had a different turn[i] value and received a gift, is it possible for the clown to not be able to go home because the while condition is still true? Give the sequence that will show this situation.**

Yes, because the children class has access to ballsNumber.

p0=getTheGift();

p0=load ballsNumber = 10

p0=ballsNumber - 1 = 9

p1=getTheGift();

p1=load ballsNumber = 10

p1=store ballsNumber = 9

p0=Store ballsNumber = 9

p0=goHome();

p1=goHome();

This can happen multiple times. If at least two processes access the ballsNumber simultaneously they will load the same value. The same value for ballsNumber is decremented twice. Since the ballsNumber is accessed by the children class and not the clown class, there is possibility for data coherence.

**Extra Credit - 6 points**

(Giving the execution sequence) that if TS is not executed automatically, the Mutual Exclusion condition with TS fails to satisfy the condition. Show where the interrupt should occur.

While(true){

.....

.....

.....

Disable Interrupts

CS

Enable Interrupts

}