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.

**(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.

**II (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] = 1p2=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

}