Name: Brandon Vo

NetId: bdv9527

CS6233 Final

Open Notes

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	16	
8	16	
9	16	
10	10	
11	10	
12	10	
13	20	
14	20	
15	20	
Bonus	2	
Total	200	

1. Consider the following C program:

```c
#include <stdio.h>
#include <unistd.h>

int main(void) {
	int i = 0;
	for (int i = 0; i <= 4; i++) {
		if (i % 2 == 0) {
			fork();
		}
		printf("foo\n");
	}
	return 0;
}
```

(a) (4 Points) How many times will "foo" be printed?

20

(b) (4 Points) How many processes will be created (including the initial process)?

8 processes will be made. At i = 0, split into two. i=0, 2 foos. i=1, two foos. At i=2, split into 4 processes. i = 2, spawn 4 foos. i=3, spawn 4 foos. At i=4, split into 8 processes, generate 8 foos.

(c) (2 Points) If `fork()` can create processes, why do we still have `exec()`?

Fork creates a new process to run the code, having both run simultaneously. Exec replaces the current program with the one specified before executing that process.

2. The dup() function will create a copy of the file descriptor. The following is the code that implements the dup() system call in xv6:

```
1  int
2  fetchint(uint addr, int *ip)
3  {
4    if(addr >= proc->sz || addr+4 > proc->sz)
5      return -1;
6    *ip = *(int*)(addr);
7    return 0;
8  }
9
10 int
11 argint(int n, int *ip)
12 {
13   return fetchint(proc->tf->esp + 4 + 4*n, ip);
14 }
15
16 static int
17 argfd(int n, int *pfd, struct file **pf)
18 {
19   int fd;
20   struct file *f;
21
22   if(argint(n, &fd) < 0)
23     return -1;
24   if(fd < 0 || fd >= NOFILE || (f=proc->ofile[fd]) == 0)
25     return -1;
26   if(pfd)
27     *pfd = fd;
28   if(pf)
29     *pf = f;
30   return 0;
31 }
32
33 int
34 sys_dup(void)
35 {
36   struct file *f;
37   int fd;
38
39   if(argfd(0, 0, &f) < 0)
40     return -1;
41   if((fd=fdalloc(f)) < 0)
42     return -1;
43   filedup(f);
44   return fd;
45 }
```

(a) (2 Points) On line 39, why do we have to use `argfd(int n, int *pfd, struct file **pf)`, instead of directly passing the value to `fd` to `sys_dup`?

This program is in kernel space which has access to all memory addresses which can be dangerous. Argfd is being used to do explicit checks to ensure that we're in the proper memory address space.

(b) (5 Points) In the `argint` function, why do we add `4 + 4*n` to `proc->tf->esp`?

The size of a stack in assembly is done in 4 bits. In order to move, the esp pointer must always be incremented in terms of 4. In addition, the esp pointer is right on top of the return in the assembly stack address, so it has to be incremented by 4 to move on top of it.

(c) (3 Points) What could go wrong if xv6 didn't have the check at line 4 of `fetchint` function?

That line checks if the address being passed is within the proper address space. This is designed to guard against malicious programs that are trying to read memory out of bounds in the kernel space. This stops user-space programs from reading memory they're not supposed to be seeing or programs that are trying to crash the kernel. Without it, someone might try to crash the kernel if written to exit the alotted space.

3. Four batch jobs, A through D, arrive at a computer center at the same time, They have estimated running times of 10, 4, 2, 7 minutes respectively. At time 3, job E arrives, which takes 3 minutes. At time 5, jobs F and G arrive, which take 6 and 5 minutes respectively. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.

(a) (5 Points) First-come, first-served (run in order of sequence: A, B, C, D, E, F, G).

Because it's first come first serve, the order is A, B, C, D, E, F, G regardless of when they arrive

10+(10+4)+(10+4+2)+(10+4+2+7)+(10+4+2+7+3-3)+(10+4+2+7+3+6-5)+ (10+4+2+7+3+6+5-5)

10+14+16+23+(26-3)+(32-5)+(37-5)=145/(all processes)

145/7=20.71

(b) (5 Points) Shortest job first.

Order: C B, (Job E Arrives), (F and G Arrive), E, G, F, D, A

2 + (2 +4) + (2+4+3-3) + (2+4+3+5-5) + (2+4+3+5+6-5) + (2+4+3+5+6+7) + (2+4+3+5+6+7+10)

2 + 6 + (9 - 3) + (14-5) + (20-5) + 27 + 37 = 102/7=14.57

4. Recall that in 32-bit x86, page directories and page tables are each made up of 1024 32-bit entries. Suppose we have 4 processes on a system, each of which has every possible virtual address mapped.

(a)(5 points) How much memory is used to store the page directories and page tables if 4KB pages are used?

1 page directory needed = 4KB

4 KB * 1024 page table entries = 4096 KB

(4KB size of directory + 4096KB page entries) * 4 processes = 16 MB needed

(b)(5 points) If 4MB pages (superpages) are used, then the entries in the page directory point directly to the page frame (i.e., no second-level page tables are used). How much memory would be taken up by page directories in this case?

No second-level page tables. Super pages point to the entries directly.

4 page tables * 4 KB = 16 KB needed.

5. (a)(2 + 4 points) What is a zombie process? Write a program that creates a Zombie.

Zombie processes are processes that have exited but still have an entry in the process table.

```
int main()
{
    if(fork() == 0)
    {
        exit(1);
    }

   while(true);
   return 0;
}
```

(b)(4 points) Explain (with diagram and a paragraph) how a priority round-robin scheduler works when the system has processes of two distinct priorities -- low priority and high priority. Work through an example with 2 high priority processes and 3 low priority processes.

One way Round-Robin can implement priority is having two different queue levels based on the priority of that queue. The 2 High priority processes would be put in their own queue while the low priority processes will be placed in their own queue.

What we can do now is allow the scheduler to put more focus on the queue containing the high priority processes. This can be done either by letting the higher priority queue run more often or by giving the processes in the higher priority queue more quanta to remain longer in the CPU.

The 2 high priority processes will be put in a queue of priority 4 to mark that it is of high priority while the 3 low priority processes will be placed in a queue of priority 1 to mark their low priority. When the Round Robin scheduler opens the priority 4 queue, it can do something like give the queue more time to stay running or loop the priority 4 queue multiple times to let the high priority processes continue running before moving down to the priority 1 queue. At the priority 1 queue, the scheduler will give the processes in this queue less quanta or do not bother looping through the queue before quickly moving back up to the high priority queue. This allows the two high priority processes to be run more often compared to the three low priority processes.

6. Below is the code for the scheduler function in xv6, which picks the next process to run. Refer to it as you answer the following questions:

```
1 // Per -CPU process scheduler.
2 // Each CPU calls scheduler () after setting itself up.
3 // Scheduler never returns . It loops , doing :
4 // - choose a process to run
5 // - swtch to start running that process
6 // - eventually that process transfers control
7 // via swtch back to the scheduler .
8 void
9 scheduler ( void )
10 {
11     struct proc *p;
12
13     for (;;){
14           // Enable interrupts on this processor .
15           sti ();
16
17           // Loop over processes table looking for process to run .
18           acquire (& ptable . lock);
19           for (p = ptable . proc ; p < $ ptable . proc [ NPROC ]; p ++){
20                 if (p-> state != RUNNABLE)
21                       continue;
22
23                 // Switch to chosen process . It is the process 's job
24                 // to release ptable . lock and then reacquire it
25                 // before jumping back to us.
26                 proc = p;
27                 switchuvm (p);
28                 p-> state = RUNNING ;
29                 swtch (& cpu -> scheduler , proc -> context );
30                 switchuvm ();
31
32                 // Process is done running for now .
33                 // It should have changed its p-> state before coming back .
34                 proc = 0;
35           }
36           release (& ptable . lock);
37
38     }
39 }
```

(a)(4 points) What scheduling algorithm is implemented by this code? Is it fair (i.e., does each process get an equal share of the CPU)?

xv6 uses round robin. Round Robin is fair because it uses quantum as a measure of how much time each process gets before moving on to the next runnable process. All processes get an equal opportunity in the CPU.

(b)(4 points) On lines 20 and 21, we skip the rest of the loop if the process state is not RUNNABLE. What is an example of another state that a process could be in? What would go wrong if we tried to run a process in that state?

If a process is in the blocked state, it's waiting on another process or event to finish before proceeding. If we execute the blocked process prematurely, it might end up missing necessary data or be unable to finish its execution, risking an error with the process.

(c)(2 points) The `swtch` function, called on line 29, is written in assembly. What is its purpose, and why does it need to be written in assembly rather than C?

The function is performing a context switch where it saves the old context before swapping to the new context. It's written in Assembly because what it's pointing to doesn't return anything since it acts as though it's self-contained.

This calls into the new context and holds the parameters of hte old context during its return. That context may change as the other process is being executed which won't be detected in C.

7. (16 points) For the following questions CIRCLE True/False. Each question is worth 2 points.

True/False (a) Throughput means the time between the moment of submission of a job and the time of its completion.

False, that's turnaround

True/False (b) Virtual memory space cannot exceed the limit of physical memory space.

False

True/False (c) The OS can suspend a process even though the process didn't yield.

True

True/False (d) A segment is of variable size and a page is of fixed size.

True

True/False (e) Threads can be used to accelerate I/O heavy processes, for example, a web server process.

False

True/False (f) In x86 the execution stack grows towards decreasing memory addresses.

True

True/False (g) Batch systems can guarantee best response time.

False

True/False (h) A user process can yield its time slice to another specific user process.

True

8. (16 points) For the following questions, circle the correct option for the answer. Each question is worth 2 points.

(a) A process, which has voluntarily given up the CPU, it should be set to B____ state.

A. Running
B. Blocked
C. Ready
D. Zombie

(b) Which scheduling algorithm allocates the CPU first to the process that requests the CPU first? A

A. first-come, first-served scheduling
B. shortest job scheduling
C. priority scheduling
D. none of the mentioned

(c) In operating system, each process has its own: D

A. address space and global variables
B. open files
C. pending alarms, signals and signal handlers
D. all of the mentioned

(d) A thread shares its resources (like data section, code section, open files, signals) with: C

A. other processes similar to the one that the thread belongs to
B. other threads that belong to the similar processes
C. other threads that belong to the same process
D. All of these

(e) Swap space exists in: B

A. primary memory
B. secondary memory
C. CPU
D. none of the mentioned

(f) In FIFO page replacement algorithm, when a page must be replaced: A

A. oldest page is chosen
B. newest page is chosen
C. random page is chosen
D. none of the mentioned

(g) A process is thrashing if: A

A. it is spending more time paging than executing
B. it is spending less time paging than executing
C. page fault occurs
D. swapping can not take place

(h) During memory allocation, external fragmentation will not occur when: D

A. first fit is used
B. best fit is used
C. worst fit is used
D. no matter which algorithm is used, it will always occur

9. (16 points) For the following questions fill in the blank with the correct answer. Each question is worth 2 points.

(a) To overcome the problem of doubling the memory access time, most virtual memory schemes make use of a special high-speed cache for page table entries called a Translation Lookaside Buffer.

(b) When instead of using one page table entry per virtual page, the system keeps one entry per physical page frame, it's called: Inverted Page table.

(c) A process in the Blocked state is waiting for user's input.

(d) Pages that have been used a lot recently will probably be used again soon, so Least Recently Used is devised as a page replacement algorithm.

(e) Program Counter contains the next instruction to execute.

(f) Scheduler are special system software that controls which process to run next.

(g) My favorite part of the course so far has been:

Learning about how we have optimized the OS for certain aspects. Scheduling and page tables.

(h) My least-favorite part of the course so far has been (besides this test):

Homework 3 when we had to write our own lottery scheduler. I had to spend a lot of time figuring out how to not create an error of locking my system too many times and understanding why certain processes were not in the scheduler.

10. Consider the following assembly program. The program starts executing at the start label. Values starting with $ are constants.
Assembly Code:

```
start:
    mov $0, %eax
    jmp two

one:
    mov $0x1234, %eax

two:
    cmp $0x1234, %eax
    je done
    call one
    mov $10, %eax

done:
    jmp done
```

(a) (5 points) What is the value of the EAX register when execution reaches the done label?

0x1234

From Start: assigns 0 to eax. Jumps to two.

At two, fails the comparison, so it gets called to one. Call has no return, so it functions as a jump.

At 1, assigns 0x1234 and moves back down to two.

At two, succeeds the comparison and jumps to done.

(b) (5 points) What is the value of AL at the same point?

AL is the lower 8 bytes. AL is 34.

11. (a) (4 points) Explain how the system timer, which triggers an interrupt at regular interrupts, allows a system with a single CPU to create the illusion that multiple processes are running simultaneously.

Interrupts divides the CPU to switch processes at regular intervals. Essentially, each process in the CPU is given a set amount of quantum to run before the interrupt is triggered and the CPU will move to the next process. This creates the illusion of hte CPU running simultaneously because the CPU is rapidly switching between multiple processes at regular intervals, juggling several processes rather than one at a time.

(b)(3 points) If there were no system timer, would it still be possible to run more than one process? What would the downsides be?

Yes, the same effect can be done by utilizing a lock. When a proces is ready to yield, it will trigger a lock and signal the other processes to enter the CPU.

However, the issue is that it's up to the programmer to write how and when a process will take the lock and when it will release the lock without creating a deadlock and avoiding starvation. In addition, this creates overhead because the process of having to context switch from one process to another while unlocking and locking a lock takes time within itself.

(c) Is there any downside to having the timer interrupt more frequently? Is so what, is it?

Timer interrupts create overhead. Everytime a timer interrupts, the process has to stop what it's doing and wait to be swapped out by the CPU. Too many timer interrupts causes too much overhead from having to constantly switch processes rather than finish the process's work.

12. (10 points) The following problem demonstrates the use of semaphores to coordinate three types of processes. Santa Claus sleeps in his shop at the North Pole and can only be woken up by either (1) all nine reindeer being back from their vacation in the South Pacific, or (2) some of the elves having difficulties making toys; to allow Santa to get some sleep, the elves can only wake him when three of them have problems. When three elves are having their problems solved, any other elves wishing to visit Santa must wait for those elves to return. If Santa wakes up to find three elves waiting at his shop's door, along with the last reindeer having come back from the tropics, Santa has decided that the elves can wait until after Christmas, because it is more important to get his sleigh ready. (It is assumed that the reindeer do not want to leave the tropics, and therefore they stay there until the last possible moment.) The last reindeer to arrive must get Santa while the others wait in a warming hut before being harnessed to the sleigh. Solve the problem using semaphores.

Here are some addition specifications:

- After the ninth reindeer arrives, Santa must invoke **prepareSleigh**, and then all nine reindeer must invoke **getHitched**.

- After the third elf arrives, Santa must invoke **helpElves**. Concurrently, all three elves should invoke **getHelp**.

- All three elves must invoke **getHelp** before any additional elves enter (increment the elf counter).

Santa should run in a loop so he can help many sets of elves. We can assume that there are exactly 9 reindeer, but there may be any number of elves.

Because prepareSleigh is the most important process, Santa decides that all elves would wait if this is invoked. In that case, all elves will be forced to wait on the Sleigh Semaphore in order to check if Santa is preparing his sleigh before asking Santa for help.

The reindeers must use their own semaphores before counting each other in order to prevent a race condition. Reindeer Semaphore is invoked alongside a reindeer counting variable to identify if a reindeer is being counted. When a reindeer returns, it takes the reindeer semaphore and activates the counting condition before entering the warming hut. Once the reindeer has counted itself and entered the hut, it will swithc the counting variable and semsignal the reindeer semaphore to let the other reindeer know that it's their turn to return.

When the 9th reindeer takes the semaphore, counts itself, enters the warming hut, and releases its semaphore, all 9 reindeer would be considered ready to go. This means Santa will wake up and prepare the sleigh. Assumign Santa isn't allowed to throw out any elves he's working with, Santa will need to check the Elf semaphore to see if it's empty before preparing the sleigh. Santa will take the elf sem(z) semaphore to stop the other elves from lining up to be counted. Santa will deal with any elves he's currently working with before throwing them out. Once the last 3 elves are dealt with, Santa will be holding the semaphore, preventing any other elves from coming inside.

Santa will wake up and invoke prepareSleigh(). Because Santa is too preoccupied to be dealing with his elves, Santa will take the elf semaphore again, but he will not be letting go for the duration of Christmas. This means the elves will not be allowed to group up and ask Santa for help because Santa is too busy.

Afterwards, all reindeers will invoke getHitched(). Assuming the reindeers need to board the sleigh one at a time, each reindeer will need to acquire their reindeer semaphore and show themselves as being counted before actually boarding the sleigh. Once that reindeer has boarded the sleigh, that reindeer will mark the counting to be done, increment the reindeer sleigh counter, and signal the reindeer semaphore to let the other reindeer know it's their turn to enter. This process repeats until the getHitched has detected that the 9th reindeer has boarded and dropped its semaphore.

Because the elves need to get into groups of three and count each other, they require an elf count variable to check their numbers. In order to prevent a race condition, elves must enter their semaphores one at a time, this is done by having a sem(x) where all elves need to line up one by one before entering groups of 3. Before they can gather up and count, they must check if Santa is holding the elf semaphore. If santa is holding the elf semaphore, it means Santa is either busy dealing with another set of elves or he is preparing his sleigh in which case, the elves must wait their turn.

If the elf semaphore is not being used and they are able to enter sem(x), that means the elves can check if they need to ask Santa for help. The elves need their own semaphore because any elves greater than 3 must wait on the first 3 elves before asking for help. The 3 elves will have to enter another elf semaphore where they can count each other to check if there are 3 elves ready. All 3 elves will invoke getHelp() at this point once they have counted.

After invoking getHelp(), Santa will be awakened. Before helpigkn the elves, Santa will grab the elf semaphore to prevent any other elf from gathering up and asking Santa for help. This wis to prevent another grou pof elves from barging in to ask for Santa for help. Then, Santa will check on the reindeer to see if the sleigh needs to be prepared. Santa will do this by taking the sleigh semaphore and counting the reindeer. If the reindeer aren't ready, then Santa drops the sleigh semaphore and consults the elves. Santa will invoke helpElves() to help the 3 elves before going back to sleep.

Once the 3 elves have been helped, the 3 elves must drop their semaphore and decrement their elf counts. To prevent a race condition, the elves must go one at a time, so a sem(z) is used to line up the elves one by one. Each elf will take the sem(z), decrement elf count then semsignal the other elf that it's his turn to go. Once the last elf decrements elf count to 0, that means he must drop sem(z) and signal the elf semaphore to let the other elves know it's their turn to group up. Then, the last elf will signal the sem(x) semaphore to let the elves know it's their turn to line up. This informs the other elves in line that they can gather up in a group of 3 and ask Santa for help.

13. (20 points) For the following questions Circle True/False. Each question is worth 2 points

T F **(A)** The size of virtual storage is limited by the actual number of main storage locations. F

T F **(B)** It is the responsibility of the operating system to control the execution of processes. T

T F **(C)** A process that is waiting for access to a critical section does not consume processor time. F

T F **(D)** It is possible for one process to lock the mutex and for another process to unlock it. F

T F **(E)** All types of UNIX files are administered by the OS by means of inodes. T

T F **(F)** The FAT File System is not kept in memory. F

T F **(G)** Two of the most widely used approaches to improve disk I/O performance are disk scheduling and disk cache. T

T F **(H)** Timestamps for a file are a reliable attribute F

T F **(I)** XV6 supports multiple processors. F

T F **(J)** Interrupts are reusable resources. F

14. (20 points) For the following questions circle the correct option for the answer. Each question is worth 4 points

(a) The four main structural elements of a computer system are: A

A) Processor, Main Memory, I/O Modules and System Bus

B) Processor, I/O Modules, System Bus and Secondary Memory

C) Processor, Registers, Main Memory and System Bus

D) Processor, Registers, I/O Modules and Main Memory

(b) A semaphore that does not specify the order in which processes are removed from the queue is a A semaphore.

A) weak

B) general

C) strong

D) binary

(c) A situation in which a runnable process is overlooked indefinitely by the scheduler, although it is able to proceed, is C.

A) mutual exclusion

B) deadlock

C) starvation

D) livelock

(d) A real-world example of D occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time.

A) mutual exclusion

B) deadlock

C) starvation

D) livelock

(e) The C unit is capable of mimicking the processor and of taking over the system bus just like a processor.

A) Interrupt Drive I/O

B) I/O Channel

C) Direct Memory Access

D) Programmed I/O

15. (20 points) For the following questions fill in the blank with the correct answer. Each question is worth 4 points.

(a) Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of main memory.

(b) An Atomic Operation is a function or action implemented as a sequence of one or more instructions that appears to be indivisible, no other process can see an intermediate state or interrupt the operations.

(c) How can the hold-and-wait condition for deadlocks be prevented?
A process must request all of its required resources at one time

(d) This mutual exclusion mechanism **doesn't** work in uniprocessor systems: Spinlock.

(e) In XV6 during an interrupt handler we would need to use a lock and also need to disable interrupts in order to avoid reentrancy.