Name: Brandon Vo

NetId: bdv9527

CS6233 Midterm

Answer the questions in the spaces provided in the question sheets. If you run out of room for an answer, you can continue on the back of the page.

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	16	
8	16	
9	14	
Total	106	

1. Consider the following C program:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(void) {
    int i = 0;
    for (i = 0; i < 4; i++) {
        if (fork() == 0) {
            printf("foo\n");
        }
        else {
            wait(NULL);
            exit(1);
        }
    }

    return 0;
}
```

(a) (5 points) How many times will foo" be printed?

4 will be printed because only one child process will actually print foo due to the if condition stating if the process is a child.

Only the child process will be printing foo. Because each process becomes a parent of a process until the end, only 4 will say foo.

(b) (5 points) How many processes will be created (including the initial process)?

1 + 2^1 + 2^2 + 2^3 = 15 processes in total
or 2^4-1 = 15 processes being made

fork() = 0 means a child process

It creates a child process then waits for it to finish. Each loop will just create 1 fork to execute then finish.

2. The following is the code that implements the kill system call in xv6:

Assembly Code:

```
1 kill :
2 mov $0x6 ,% eax
3 int $0x40
4 ret
```

C code:

```
1 // Fetch the int at addr from the current process .
2 int
3 fetchint ( uint addr , int *ip)
4 {
5   if( addr >= proc ->sz || addr +4 > proc ->sz)
6       return -1;
7   *ip = *( int *)( addr );
8   return 0;
9 }
10
11 // Fetch the nth 32- bit system call argument .
12 int
13 argint (int n, int *ip)
14 {
15  return fetchint (proc ->tf ->esp + 4 + 4*n, ip );
16 }
17
18 int
19 sys_kill ( void )
20 {
21  int pid ;
22
23  if( argint (0, &pid ) < 0)
24      return -1;
25  return kill (pid );
26 }
```

(a) (3 points) On line 15 of the C code, in the argint function, why do we add 4 to proc->tf->esp?

Because calling esp alone puts you on the return stack of the previous call frame. You need to move up 4 bits to reach past the return address.

(b) (2 points) On that same line, why do we multiply n by 4?

Because size and bits used in assembly are of the size 4 because of 4 bits, so we always move in terms of 4 bits.

Because n is trying to retrieve data from the stack frame, it have to move in 4's to move from one address register to another.

(c) (5 points) What could go wrong if xv6 didn't have the check at line 5 in fetchint?

It checks to see if the memory address being pointed at is in the proper memory space. Without that check, you wouldn't be able to tell if you were out of bounds or in the wrong address space. In xv6, this could risk accessing wrong data or breaking memory being used by another process.

3. Three batch jobs, A through C, arrive at a computer center at the same time. They have estimated running times of 10, 6, 2 minutes respectively. At time 3, jobs D and E arrive, which take 4 and 8 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 A, B, C, D, E).

A, B, C, D, E
Time 0:
A - 10 minutes
B - 6 minutes
C - 2 minutes

Time 3:

D - 4 minutes
E - 8 minutes

10+16+18 + (22-3) + (30-3) = 90 minutes
90/5=18 average turnaround time

(b) (5 points) Shortest job first.

Order at time 0 would be C, B, A. Order at time 3 would be D, E added

2 + 8 + 18 + (22 - 3) + (30-3) = 74 minutes
74/5 = 14.8 average turnaround time

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?

Using the entire virtual memory address space with 4 KB pages.
4 processes meaning 4 page frames being used.
1024 = 4MB.

2^32 / 4KB pages = 2^16 pages
2^16/4MB or 2^16 pages / 1024 entries = 2^6 page tables being used

(2^32 / 4MB) * sizeof(4 page tables because 4 processes)
(2^32 / 4MB) * 4KB worth of page tables being used
4*(2^32 / 4 MB) * 4KB worth of memory being used for 4 processes.

(b) (5 points) If 4MB pages (super pages) 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?

Super pages point directly towards the 4MB page frame.

2^21 or 4 MB for the super pages.
4 processes meaning 4 directories.

(4 GB / 4MB) = 1024 page directory entries for one level pages

4*(4GB / 4MB) worth of memory being taken up for 4 processes.

5. Below is the code for `swtch()` from xv6:

```
1 # Context switch
2 #
3 # void swtch ( struct context ** old , struct context *new);
4 #
5 # Save current register context in old
6 # and then load register context from new.
7
8 .globl swtch
9 swtch :
10 movl 4(% esp ), %eax
11 movl 8(% esp ), %edx
12
13 # Save old callee - save registers
14 pushl %ebp
15 pushl %ebx
16 pushl %esi
17 pushl %edi
18
19 # Switch stacks
20 movl %esp , (% eax )
21 movl %edx , %esp
22
23 # Load new callee - save registers
24 popl %edi
25 popl %esi
26 popl %ebx
27 popl %ebp
28 ret
```

(a) (5 points) How is the program counter (%eip) saved and restored by this code?

When the call has been fully executed and finished, it will automatically put the %eip back into the stack and restore the %eip.

Line 10 and line 11 calls the stack needed to execute the process.
Then, the old registers are saved.
Line 20 and line 21 switches the stack and saves the old stack ESP pointer in EAX.

(b) (5 points) Give an example of one other piece of program state that is not explicitly saved by this code and explain why that doesn't cause problems when performing a context switch.

The base registers like eax, ecx, and edx are not saved because they are implicitly preserved by the swtch() function. When the old stack frame is restored, these registers will be restored due to being a part of the the kernel context which itself gets restored through the context switch.

6. On the next page 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:

(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)?

Scheduling algorithm appears to be the round robin scheduling policy as it cycles through every process to see if there is one that needs to run and transfers control to the next.

Round Robin is fair as it gives every process a chance to run based on quantum issued out.

(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, that means it's still waiting on something to finish or some form of data to come in. It can't be executed at the moment otherwise it will be incomplete or unable to finish its execution without the necessary data. Could risk a crash.

(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?

This line is saving the old context and the new context of the context being changed. It's 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 the old context during return. That context may change as the other process is being executed which won't be detected in C.

```
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 process 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          switchkvm ();
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 }
```

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

False T F (a) A process that is not in main memory is immediately available for execution, regardless of whether or not it is awaiting an event.

True T F (b) For virtual memory, a hardware mechanism is needed for translating relative addresses to physical main memory addresses at the time of execution of the instruction that contains the reference.

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

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

True T F (e) The OS may suspend a process if it detects or suspects a problem.

True T F (f) If a system does not employ virtual memory each process to be executed must be fully loaded into main memory.

True T F (g) An example of an application that could make use of threads is a browser.

False T F (h) Segmentation is not visible to the programmer.

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

(a) Main memory divided into a number of equal size frames is the A technique.

A) simple paging
B) dynamic partitioning
C) fixed partitioning
D) virtual memory segmentation

(b) The chunks of a process are known as A .

A) pages
B) addresses
C) frames
D) segments

(c) Available chunks of memory are known as A .

A) frames
B) segments
C) addresses
D) pages

(d) 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

(e) The D holds the address of the next instruction to be fetched.

A) Accumulator (AC)
B) Instruction Register (IR)
C) Instruction Counter (IC)
D) Program Counter (PC)

(f) Instruction processing consists of two steps: A

A) fetch and execute
B) instruction and execute
C) instruction and halt
D) fetch and instruction

(g) The processor itself provides only limited support for multiprogramming, and C _____ is needed to manage the sharing of the processor and other resources by multiple applications at the same time.

A) memory
B) data
C) software
D) hardware

(h) When one process spawns another, the spawned process is referred to as the B _____ .

A) zombie process
B) child process
C) stack process
D) parent process

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

(c) A process in the Ready state is in main memory and available for execution.

(d) When the system spends most of its time swapping pieces rather than executing instructions it leads to a condition known as Thrashing .

(e) When instead of using one page table entry per virtual page, the system keeps one entry per physical page frame, i called: Inverted Page tables

(f) My **favorite** part of the course so far has been:
Studying the methods used to optimize the OS.

(g) My **least**-favorite part of the course so far has been (besides this test):
Studying assembly, especially the part where we had to translate C code into assembly.