Akash Lankala
CS 354
Lab 1
February 6th, 2020

Using ONE Late Day

Problem 3.1

(a)
How many processes can exist in XINU?

In process.h, NPROC is defined as:

```
process.h:#define NPROC       8
```

In conf.h, NPROC is defined as:

```
conf.h:#define    NPROC      100    /* number of user processes        */
```

In process.h, it is commented that the definition of NPROC is the maximum number of processes in the system, which is different from the maximum number of user processes.

We are looking for the maximum number of processes that can exist in XINU, which appears to be 8.

(b)
Locate where the constant NULLPROC is defined and determine its value.

In process.h, NULLPROC is defined as:

```
process.h:#define NULLPROC    0     /* ID of the null process          */
```

The value of NULLPROC is 0, which is also the ID of the null process, as stated.

Determine the C type of the priority field in the process table of XINU. Describe the procedure you followed to arrive at the answer. Specify the maximum and minimum priority values, at least in principle, that a XINU process may take. Where in this range does the priority of the NULL process fall?

Based on the variable definition INITPRIO in process.h, it appears that the range of priority values are between 0-20.

```
process.h:#define INITPRIO    20    /* Initial process priority        */
```

The maximum priority value is 20 and minimum is 0. In main(), the shell is created with a priority of 50.

Akash Lankala
CS 354
Lab 1
February 6th, 2020

The priority of the NULL process has a value of 0 as seen in this statement in process.h:

```
process.h:#define NULLPROC    0     /* ID of the null process          */
```

(c)
Make changes to initialize.c

View the changes made in initialize.c.

Problem 3.2
Remove "Hello World!" write a function void mymotd(void) in mymotd.c which outputs using kprintf().

View changes made to main() and mymotd.c.

Problem 3.3
(a)
Code a modified version of create(), rcreate(), same arguments as create(), which puts the newely created process in ready state.

Check rcreate() for modified version of create().

(b)
In Linux, which runs first: child or parent? Experimentally gauge an answer to the question by writing and testing code using fork() and write().

The parent runs first. This was discovered experimentally from writing test code in *parentchild.c* in lab1/. Here, pid was used to keep track of process ID, fork() was used to create the parent and child process, and a few statements were printed using the write() system call. It was seen that the parent prints out to the terminal first.

In the rcreate() version of create() in XINU, which process will run first after rcreate() is called when the process running main() is created in 3.3(a)?

Rcreate() calls interrupt mask, initializes process table entry for new processes, and the initial state is suspended.
The interrupt mask process is called first in rcreate() after the process running main() is created. Ready() is also called from rcreate().

Akash Lankala
CS 354
Lab 1
February 6th, 2020

Problem 3.4

Modify trap() to print your username, name, and the XINU system variable clktime. Trigger interrupt 0 by performing divide by zero at the beginning of main(). Confirm that the system reaches the code in trap().

Check trap() function and main() function for changes. Some are commented out, such as the divide by zero at the beginning of main().

Output:

```
Xinu for galileo -- version #23  (alankala)  Thu Feb  6 19:54:47 EST 2020
------------------------------------------------------------------------
Ethernet Link is Up
MAC address is 98:4f:ee:00:15:07
250076704 bytes of free memory.  Free list:
  [0x0015C1E0 to 0x0EFD8FFF]
  [0x0FDEF000 to 0x0FDEFFFF]
   111505 bytes of Xinu code.
  [0x00100000 to 0x0011B390]
   135784 bytes of data.
  [0x0011F060 to 0x001402C7]


Akash Lankala
Username: alankala
Name: Akash Lankala
clktime: 0.0
Xinu trap!
exception 0 (divided by zero) currpid 2 (Main process)
CS 1000008 eip 1028A0
eflags 10256
register dump:
eax 00000008 (8)
ecx 00000000 (0)
edx 00000000 (0)
ebx 00121000 (1183744)
esp 0EFD8FC0 (251498432)
ebp 0EFD8FC0 (251498432)
esi 00000000 (0)
edi 00000000 (0)
```

Akash Lankala
CS 354
Lab 1
February 6th, 2020

panic: Trap processing complete...

Explain what happens to XINU at the end dividing by zero in an app generates interrupt 0.

Assembly code in intr.S under the label _Xint0 has 7 lines of code, the last line calls Xtrap. The first line of Xtrap calls the trap function trap() in evec.c. This is a function call from assembly code to c code. The trap() function in evec.c takes in two arguments: int inum (interrupt number), and long *sp (saved stack pointer).
It displays the message, and disable() within panic.c disables the interrupts.

Explain how passing of the two arguments is accomplished following the CDECL convention.
The C parameters are passed to the called routine by pushing them onto the stack. They are pushed onto the stack from right to left.

Problem 3.5

Determine the name of the function that XINU sets up as the function that calls main(). Look at the code of this virtual called in system/ and determine what it does.

INITRET is the function that XINU sets up as the function that calls main(). In process.h, INITRET is defined as the address to which process returns. In *create.c*, INITRET pushes on the return address.

In the special case where the process that terminates is the last process outside the NULL process, determine what happens by continuing to follow XINU's source code. In the chain of functions being called starting with the virtual function to which main() returns, what is the last function that is called, where is its source code, and what does it do?

The last function that is called is restore() in create.c. It returns to the main().

In environments where power consumption is an important issue, is XINU's approach to "shutting down" after the last process has terminated a good solution? What might be a better solution? Explain your reasoning.

Shutting down is a good solution but decreasing the number of processes running at a time may also lower power consumption in environments where that is necessary. Currently, XINU can

handle a certain number of processes, so perhaps decreasing that level to meet certain energy standards or power savings may be a possible solution to this.

Bonus Problem

What does it mean by saying fork() is special in that it is the only system call that "returns twice."?
When we use fork(), parent and child, child returns PID, parent returns 0. Thus, it returns twice.

Why does this feature of fork() not apply to XINU's create()
In XINU's create, the creator does nothing. Creator will turn itself into do-nothing null process. In execve, the child inherits the parents process. XINU's create() creates a new process. XINU's create() will create a new PID. Whereas in execve it does not do that, it replaces the caller.

We say that Linux/UNIX execve() system call is special in that it is the only system call that does not return when it succeeds. What does this mean? Why does it not apply to XINU's create?
Execve does not create a new process, it replaces the called process, and that is why it never returns.