Gray Houston
CS 354
Dr. Park
Using 1 Late Day

Lab 2

3.) Hijacking a Process through Stack Smashing

My method for hacking the victim process was to use the process table to locate the stack of the victim process within memory. Because it is guaranteed in the question that the victim process is created immediately before the attacker process, the PID of the victim will simply be currpid – 1. By knowing the victim's PID, we can access the information about it in its process table entry.

Then, we can get the value of the stack base of the victim from the victim's process table entry. From there, it is relatively trivial to find the return address of the sleepms() function call made by the victim process.

By smashing the return address of the call to sleepms(), not only is the malware function able to run, but the malware is also able to return to the victim's calling function. When my code runs, not only does the malware code change the value of the myvictimglobal variable, but the myvictim() function will eventually finish its execution and print the altered myvictimglobal value.

4.) Monitoring CPU Usage of Processes

In this section, I implemented the functionality so that the OS keeps track of the amount of CPU time that a process uses. I added a column entry to the process table named *prcpuused*, which is an unsigned integer which represents the number of milliseconds the process has run on the CPU. This value is updated by the clock interrupt handler. Every millisecond, when the clkhandler runs, the *prcpuused* value of the currently running process is incremented by 1. Therefore, I do not need to use a secondary variable to determine the CPU usage time of a process. The only process whose *prcpuused* value is not updated is the null process because of the setup of the fair scheduler (this will be explained in the next question).

For this to work, I needed to update initialize.c so that the null process had a starting value. I also changed create.c so that all created processes would have an initial *prcpuused* value of 1.

In regards to the test cases from the previous lab, all the process eventually used the same amount of CPU time. However, the processes with a declared higher priority ran earlier than the other printloop processes.

To be able to see how much time has been spent by every process, I have altered the *ps* shell command so that the CPU time is one of the printed fields. Therefore, you can see the running time and effectively the priority of all running processes.

5.)

For the fair scheduler, I decided to completely disregard the *prprio* value stored in the process table and only use the *prcpuused* value to determine priority. Processes that have used the CPU less will gain priority over those that have used it more. I changed the workings of the ready-process list so that it

stores processes in a minimum priority queue such that the first element is the ready process with the lowest *prcpuused* value. Additionally, I initialize the null process' *prcpuused* field with the maximum int value so that the null process always has the lowest priority of any ready process. Additionally, I altered the scheduler in such a way that the round-robin policy for processes of equal priority (i.e. processes who have used the same amount of CPU time) is still enforced. It should also be noted that I changed the functionality of kprintf. Because many of the context switches were occurring while text was being printed to the screen and the information was effectively useless, I altered kprintf so that context switches were disabled during the printing. Because of this, all of my testing output is meaningful.

In regards to the first test case, with all CPU intensive processes, it is obvious that the scheduler is running the processes in a fair manner. The processes effectively run iteratively in the order in which they were created, since once process A executes, it has used more time than process B and will be context switched off.

In regards to the second test case, with all I/O intensive processes, it is obvious that the scheduler is fair. All the processes run in a round-robin fashion which is trademark of processes receiving equal amounts of time.

For the third test case, where half the processes are CPU intensive and the other half are I/O intensive, the output indicates that the scheduler is running in a fair manner. While the I/O processes are sleeping, the CPU processes run in what is effectively a round-robin fashion. Once the I/O processes wake up from their sleep, they are context switched on and run. This can be seen in the testing output.

At the end of this document is the output of my tests if you wish to read it. You can get this output by running XINU with my main method.

Bonus.)
As stated in the handout, there is a serious potential problem that newer processes will gain significant priority over older processes simply because of when the processes were created. A fair way to solve this problem is to periodically “forget” how much time all running processes have used the CPU and all values of *prcpuused* would be reset to 1. Therefore, newer processes would only hold an advantage for a small period of time before the process history is forgotten. Additionally, this system of periodically forgetting process history should work because it is Linux's answer to this problem and has been shown to be very effetive.

Additionally, this system would work well for processes that change between I/O bound and CPU bound. If a process changed from CPU to I/O bound, once all process history has been forgotten, this process will gain a higher priority to account for its I/O bound state.

Fair Scheduler Test Output

------- CPU-Intensive Muthafuckas -------

PID = 9 CPU-Intensive "cpu1"
Outer Loop = 0

Priority = 1/63
Time Slice Remaining = 8

PID = 10 CPU-Intensive "cpu2"
Outer Loop = 0
Priority = 1/63
Time Slice Remaining = 8

PID = 11 CPU-Intensive "cpu3"
Outer Loop = 0
Priority = 1/63
Time Slice Remaining = 8

PID = 12 CPU-Intensive "cpu4"
Outer Loop = 0
Priority = 1/63
Time Slice Remaining = 8

PID = 9 CPU-Intensive "cpu1"
Outer Loop = 1
Priority = 1/127
Time Slice Remaining = 4

PID = 10 CPU-Intensive "cpu2"
Outer Loop = 1
Priority = 1/127
Time Slice Remaining = 4

PID = 11 CPU-Intensive "cpu3"
Outer Loop = 1
Priority = 1/127
Time Slice Remaining = 4

PID = 12 CPU-Intensive "cpu4"
Outer Loop = 1
Priority = 1/127
Time Slice Remaining = 4

PID = 9 CPU-Intensive "cpu1"
Outer Loop = 2
Priority = 1/190
Time Slice Remaining = 1

PID = 11 CPU-Intensive "cpu3"
Outer Loop = 2
Priority = 1/190
Time Slice Remaining = 1

PID = 10 CPU-Intensive "cpu2"

Outer Loop = 2
Priority = 1/191
Time Slice Remaining = 10

PID = 12 CPU-Intensive "cpu4"
Outer Loop = 2
Priority = 1/191
Time Slice Remaining = 10

PID = 9 CPU-Intensive "cpu1"
Outer Loop = 3
Priority = 1/253
Time Slice Remaining = 8

PID = 10 CPU-Intensive "cpu2"
Outer Loop = 3
Priority = 1/255
Time Slice Remaining = 6

PID = 11 CPU-Intensive "cpu3"
Outer Loop = 3
Priority = 1/253
Time Slice Remaining = 8

PID = 12 CPU-Intensive "cpu4"
Outer Loop = 3
Priority = 1/255
Time Slice Remaining = 6

PID = 9 CPU-Intensive "cpu1"
Outer Loop = 4
Priority = 1/317
Time Slice Remaining = 4

PID = 9: Total CPU Time Used = 318

PID = 10 CPU-Intensive "cpu2"
Outer Loop = 4
Priority = 1/319
Time Slice Remaining = 2

PID = 10: Total CPU Time Used = 320

PID = 11 CPU-Intensive "cpu3"
Outer Loop = 4
Priority = 1/318
Time Slice Remaining = 3

PID = 11: Total CPU Time Used = 319

PID = 12 CPU-Intensive "cpu4"
Outer Loop = 4
Priority = 1/319
Time Slice Remaining = 2

PID = 12: Total CPU Time Used = 320

------- I/O-Intensive Muthafuckas -------

PID = 13 I/O-Intensive "io1"
Outer Loop = 0
Priority = 1/1
Time Slice Remaining = 10

PID = 14 I/O-Intensive "io2"
Outer Loop = 0
Priority = 1/1
Time Slice Remaining = 10

PID = 15 I/O-Intensive "io3"
Outer Loop = 0
Priority = 1/1
Time Slice Remaining = 10

PID = 16 I/O-Intensive "io4"
Outer Loop = 0
Priority = 1/1
Time Slice Remaining = 10

PID = 13 I/O-Intensive "io1"
Outer Loop = 1
Priority = 1/2
Time Slice Remaining = 10

PID = 14 I/O-Intensive "io2"
Outer Loop = 1
Priority = 1/2
Time Slice Remaining = 10

PID = 15 I/O-Intensive "io3"
Outer Loop = 1
Priority = 1/2
Time Slice Remaining = 10

PID = 16 I/O-Intensive "io4"
Outer Loop = 1
Priority = 1/2
Time Slice Remaining = 10

PID = 13 I/O-Intensive "io1"
Outer Loop = 2
Priority = 1/3
Time Slice Remaining = 10

PID = 14 I/O-Intensive "io2"
Outer Loop = 2
Priority = 1/3
Time Slice Remaining = 10

PID = 16 I/O-Intensive "io4"
Outer Loop = 2
Priority = 1/3
Time Slice Remaining = 10

PID = 15 I/O-Intensive "io3"
Outer Loop = 2
Priority = 1/3
Time Slice Remaining = 10

PID = 13 I/O-Intensive "io1"
Outer Loop = 3
Priority = 1/4
Time Slice Remaining = 10

PID = 14 I/O-Intensive "io2"
Outer Loop = 3
Priority = 1/4
Time Slice Remaining = 10

PID = 16 I/O-Intensive "io4"
Outer Loop = 3
Priority = 1/4
Time Slice Remaining = 10

PID = 15 I/O-Intensive "io3"
Outer Loop = 3
Priority = 1/4
Time Slice Remaining = 10

PID = 13 I/O-Intensive "io1"
Outer Loop = 4
Priority = 1/5
Time Slice Remaining = 10

PID = 13: Total CPU Time Used = 6

PID = 16 I/O-Intensive "io4"

Outer Loop = 4
Priority = 1/5
Time Slice Remaining = 10

PID = 16: Total CPU Time Used = 6

PID = 14 I/O-Intensive "io2"
Outer Loop = 4
Priority = 1/5
Time Slice Remaining = 10

PID = 14: Total CPU Time Used = 6

PID = 15 I/O-Intensive "io3"
Outer Loop = 4
Priority = 1/5
Time Slice Remaining = 10

PID = 15: Total CPU Time Used = 6

------- Half & Half Muthafuckas -------

PID = 17 CPU-Intensive "cpuA"
Outer Loop = 0
Priority = 1/64
Time Slice Remaining = 8

PID = 18 CPU-Intensive "cpuB"
Outer Loop = 0
Priority = 1/64
Time Slice Remaining = 8

PID = 19 I/O-Intensive "ioA"
Outer Loop = 0
Priority = 1/1
Time Slice Remaining = 10

PID = 20 I/O-Intensive "ioB"
Outer Loop = 0
Priority = 1/1
Time Slice Remaining = 10

PID = 17 CPU-Intensive "cpuA"
Outer Loop = 1
Priority = 1/127
Time Slice Remaining = 9

PID = 18 CPU-Intensive "cpuB"
Outer Loop = 1

Priority = 1/129
Time Slice Remaining = 9

PID = 17 CPU-Intensive "cpuA"
Outer Loop = 2
Priority = 1/191
Time Slice Remaining = 9

PID = 18 CPU-Intensive "cpuB"
Outer Loop = 2
Priority = 1/193
Time Slice Remaining = 9

PID = 19 I/O-Intensive "ioA"
Outer Loop = 1
Priority = 1/2
Time Slice Remaining = 10

PID = 20 I/O-Intensive "ioB"
Outer Loop = 1
Priority = 1/2
Time Slice Remaining = 10

PID = 17 CPU-Intensive "cpuA"
Outer Loop = 3
Priority = 1/255
Time Slice Remaining = 8

PID = 18 CPU-Intensive "cpuB"
Outer Loop = 3
Priority = 1/258
Time Slice Remaining = 8

PID = 19 I/O-Intensive "ioA"
Outer Loop = 2
Priority = 1/3
Time Slice Remaining = 10

PID = 20 I/O-Intensive "ioB"
Outer Loop = 2
Priority = 1/3
Time Slice Remaining = 10

PID = 17 CPU-Intensive "cpuA"
Outer Loop = 4
Priority = 1/319
Time Slice Remaining = 8

PID = 17: Total CPU Time Used = 320

PID = 18 CPU-Intensive "cpuB"
Outer Loop = 4
Priority = 1/324
Time Slice Remaining = 9

PID = 18: Total CPU Time Used = 325

PID = 19 I/O-Intensive "ioA"
Outer Loop = 3
Priority = 1/4
Time Slice Remaining = 10

PID = 20 I/O-Intensive "ioB"
Outer Loop = 3
Priority = 1/4
Time Slice Remaining = 10

PID = 19 I/O-Intensive "ioA"
Outer Loop = 4
Priority = 1/5
Time Slice Remaining = 10

PID = 19: Total CPU Time Used = 6

PID = 20 I/O-Intensive "ioB"
Outer Loop = 4
Priority = 1/5
Time Slice Remaining = 10

PID = 20: Total CPU Time Used = 6