CS 354 Spring 2024

Lab 1 Part-2: Getting Acquainted with XINU [75 pts]

Due: 1/26/2024 (Friday), 11:59 PM

1. Objectives

The objectives of this lab assignment is to familiarize you with the steps involved in compiling and running XINU in our lab, simple features of XINU processes and creating a few XINU processes.

---

2. Readings

- Chapters 1, 2 from the XINU textbook.

Please use a fresh copy of XINU, xinu-spring2024.tar.gz, but for preserving the greetings() function from lab1 part-1 and removing all code related to xsh from main() (i.e., you are starting with an empty main() before adding code to perform testing). As noted before, main() serves as an app for your own testing purposes. The GTAs when evaluating your code will use their own main() to evaluate your XINU kernel modifications.

---

3. Process management [75 pts]

(a) *Process creation continued:* Code a new XINU system call, createvariant(), that has the same function prototype as create() but behaves slightly differently:

pid32 createvariant(void *funcaddr, uint32 ssize, pri16 priority, char *name, uint32 nargs, ...);

First, if the value of the second argument of createvariant() is less than NULL process stack size, the newly created process's stack size should be set to the size of the stack

of the NULL process. As mentioned in lab1 part-1 pdf, sysinit() sets up the NULL process and there you can see that its stack size is set to NULLSTK (which is defined in one of the header files in include/). Second, if the value of the third argument of createvariant() is 0, the newly created process's priority is set to the parent process priority plus 1. Otherwise, priority is set to the third argument that has been passed.

create() remembers the parent PID of a newly created process in process table field, pid32 prparent. createvariant() checks the value of the third argument and if it is 0, it uses the parent process's priority to set the child process priority. Code createvariant() in createvariant.c under system/. Compile a new XINU binary xinu.xbin on a frontend machine. Use main() in main.c as your application layer test code to assess correct operation of createvariant() as noted in lab1 part-1 pdf. The TAs will use their own main.c to test your kernel modification.

(b) *Get parent process state system call.* Code a new XINU system call, int getparentprstate(pid32), in system/getparentprstate.c that uses the prparent process table field to return the parent process's state of the process specified in the argument. getparentprstate() must also check that the specified PID falls within the valid range of PIDs 0, ..., NPROC-1 and the PID is assigned to any one of the processes. If the argument passed is invalid, getparentprstate() returns SYSERR. Annotate your code, test and verify that it works correctly.

(c) *Process termination.* We will consider what it means for a process to terminate in XINU. There are two ways for a process to terminate. First, a process calls exit() which is a wrapper function containing system call kill(), kill(getpid()). In the relevant part of kill() the current process (i.e., process in state PR_CURR executing on Galileo's CPU) removes itself from the process table and calls the XINU scheduler, resched(), so that it may select a ready process to run next. As one might expect, XINU selects the highest priority ready process that is at the front of the ready list to run next. The ready list is a priority queue sorted in nonincreasing order of process priority. Since XINU's idle process is always ready when it is not running (it only runs when there are no other ready processes since it is configured to have strictly lowest priority), the ready list is never empty unless the idle process executes on the CPU. Other chores carried out by kill() include closing descriptors 0, 1, 2, and freeing the stack memory of the terminating process.

Second, a process terminates "normally" without calling exit() (i.e., kill()). Since the first argument of create() is a function pointer, normal termination implies returning from create() to its caller by executing the ret instruction in x86. The question is where does create() return to since it was not called by another function. For example, if a process calls create(main, 1024, 20, "main", 0, NULL) then a child process is created which executes the code of function main(). When main() executes ret where does it return to? The technique used by XINU is to set up the run-time stack upon process creation so that it gives the illusion that another function called create(). This fake caller is identified by the macro INITRET whose definition can be found in a header file in include/. Playing detective, trace the sequence of events that transpire when a process terminates normally by create() executing the x86 ret instruction.

Code a new XINU system call, void returnwrapper(void), in system/returnwrapper.c that outputs a message “Process terminating” and then calls kill(getpid()) to terminate the current process.

Define a new macro INITRET2 in include/process.h and define it as a function pointer of returnwrapper similar to how INITRET is defined as a function pointer of userret.

Code a new XINU system call, createvariant2(), that has the same function prototype as create() but behaves slightly differently:

pid32 createvariant2(void *funcaddr, uint32 ssize, pri16 priority, char *name, uint32 nargs, ...);

createvariant2() should set the return address of the process to INITRET2 instead of INITRET. When a process created using createvariant2() terminates “normally” without calling exit(); should output the message “Process terminating” before terminating.

Code createvariant2() in createvariant2.c under system/. Use main() in main.c as your application layer test code to assess correct operation of createvariant2() as noted in lab1 part-1 pdf. The TAs will use their own main.c to test your kernel modification.

Important: When new functions including system calls are added to XINU, make sure to add its function prototype to include/prototypes.h. The header file prototypes.h is included in the aggregate header file xinu.h. Every time you make a change to XINU, be it operating system code (e.g., system call or internal kernel function) or app code,

you need to recompile XINU on a frontend Linux machine and load a backend with the new xinu.xbin binary.

Turn-in instructions

General instructions:

When implementing code in the labs, please maintain separate versions/copies of code so that mistakes such as unintentional overwriting or deletion of code is prevented. This is in addition to the efficiency that such organization provides. You may use any number of version control systems such as GIT and RCS. Please make sure that your code is protected from public access. For example, when using GIT, use git that manages code locally instead of its on-line counterpart github. If you prefer not to use version control tools, you may just use manual copy to keep track of different versions required for development and testing. More vigilance and discipline may be required when doing so.

Submission format for Lab 1 Part-2:

```
lab1-part2

|--- xinu-spring2024
```

Specific instructions:

1. Format for submitting kprintf() added for testing and debugging purposes in kernel code:

- For problems where you are asked to print values using kprintf(), use conditional compilation (C preprocessor directives #define combined with #ifdef and #endif) with macro XINUTEST (in include/process.h) to effect print/no print depending on if XINUTEST is defined or not. For your debug statements, do the same with macro XINUDEBUG.

2. Please make sure to follow the file and function naming convention as mentioned in the lab handout.

3. Electronic turn-in instructions:

i) Go to the xinu-spring2024/compile directory and run "make clean".

ii) Go to the directory where lab1-part2 (lab directory containing xinu-spring2024/) is a subdirectory.

For example, if /homes/alice/cs354/lab1-part2/xinu-spring2024 is your directory structure, go to /homes/alice/cs354

iii) Type the following command

```
turnin -c cs354 -p lab1-part2 lab1-part2
```

You can check/list the submitted files using

```
turnin -c cs354 -p lab1-part2 -v
```

Please make sure to disable all debugging output before submitting your code.