American University in Cairo (AUC)
Computer Science and Engineering Department
Operating Systems
CSCE-345/3401
Fall 2020
Assignment 3 - Redundant Fork for Fault Tolerance

Assigned: Wednesday, September 30th, 2020

Due: Wednesday, October 21st, 2020 at 10:00 am

Purpose

In this assignment you are required to modify the Linux kernel. This assignment is and individual assignment that you should work on it on your own and is designed to enrich your kernel development skills. In this assignment you will need to read parts for the kernel code, understand it, and modify/add to it to achieve the target goal. In this assignment you will add a number of system call to achieve some fault tolerance mechanism that will allow a killed process to resume execution under certain conditions; simply you will be implementing a new version of the ***fork system call*** to provide redundancy and fault tolerance.

Basics

In this assignment you will have to add a number of new system calls; one main important system call and a couple of another two helper system calls. The main system call you need to add is the ***pfork*** system call. The ***pfork*** system call is like the ***fork*** system call, but it is not :)

The **pfork** system call should fork two children processes instead of one. One child process should be in a runnable state and on the run queue, while the other one should be suspended state on a wait queue. From now on, we will refer to the runnable one as ***active*** and the other suspended one as ***standby***. As long as the ***active*** process is running the ***standby*** will stay in the suspended state. As soon as the ***active*** one is terminated the ***standby*** should start executing and becomes active.

All your code should be submitted in the form of one or more patches. You should learn how to create a patch for multiple recursive directories. This will entail taking a copy of

the directories you are going to work on before starting to apply your amendments and use the ***diff*** command to create the patch(es). It is worthy reading the man page of the ***diff*** command.

Details

Your kernel amendments should achieve the following as a minimum:

1. You should extend the process block data structure to contain extra fields as a minimum (you might need to add more fields to achieve your target):
    1. **pfork_standby_pid:** this field should be set in the active process to contain the pid of the standby one, and it should only be set to a value other than zero only if ***pfork*** is used.
    2. **pfork_active_pid:** this field should be set in the standby process to contain the pid of the active one, and it should only be set to a value other than zero only if ***pfork*** is used.
    3. **pfork_status:** a status value that can be set through a system call.
2. Add a new system call and name it ***pfork***.
3. ***pfork*** should perform the following:
    1. Fork two child processes instead of one.
    2. Put the first one on the active run queue.
    3. Put the second one in the wait queue and store in its pfork_active_pid the pid of the active child.
4. Add the following system calls:
    1. **set_pfork_status:** sets the value of the pfork_status to some value based on the program logic. When invoked from the active process it will set the pfork_status of both the active and standby processes.
    2. **get_pfork_status:** returns the value of the pfork_status.
    3. **get_pfork_sibling_pid:** return pfork_standby_pid when invoked by the active process and pfork_active_pid when invoked by the standby process.
    4. **pfork_who**: returns 1 when invoked from the active process, 2 when invoked from the standby, and 0 when invoked from a process that was not created by the pfork system call.
5. There are set of ***SYSCALL_DEFINE*** macros that you should learn and read about to be able to add new systems calls with the right number of arguments.
6. You will also need to learn about some kernel APIs such as **find_get_task_by_vpid**, **task_tgid_vnr**, **do_fork**, and more.
7. When the active child terminals for any reason the standby process should be unsuspended and start execution; based on the code of the process and how it utilize the ***pfork_status*** execution should be resumed, Please refer to the example code presented at the end of the assignment document.
8. You will essentially need to hook the exit system call and any termination signals that are applicable.
9. You are required to build a user side library and call it ***pfork*** (***pfork.h*** and ***pfork.c***) which contains all the wrapping functions for the above required

system calls. The function wrappers should utilize the ***syscall*** library function to invoke kernel level system calls from the user space.

10. Some of main files that you should read, understand, and modify are ***core.c***, ***sched.h***, ***fork.c***, and more.
11. You will need to change a number of make files as well as creating some new make files.

Make sure you setup you development environment as follows:

1. First of all you will need to get the kernel source. For consistency we will get for you the kernel source version that you should work on so we have a common version for the kernel source that will be used by all the students. Please download it from here: https://drive.google.com/drive/folders/1UxqWbnf4FO9a4P85Xn9idhfb8kgzEYFB?usp=sharing
2. Moreover you should build a virtual machine on VirtualBox and install on it the latest linux distribution that you prefer, I use mint; **this is a must**.
3. Make sure you install all needed packages for kernel development and update your environment to the latest using apt-get or similar package management tools; you should be able to dig for that.
4. Copy the downloaded kernel and save it inside your virtual machine.
5. You need to go through one round of compiling the downloaded kernel successfully and booting from it before you start working and modifying code.
6. Make sure that you make a copy of each directory you modify files in so you can generate a patch.

Deliverables

Everything should be submitted to Blackboard. Include a Readme file to explain anything unusual to the TA. Your kernel code must be included in one or more patch files. Other associated filed, such as the user space library pfork code should be included in .c and.h files and stored under a separate director. A test program that utilizes the ***pfork*** system call and the other required system calls which demonstrate the required functionalities should also be included. Create a single directory that include your patches, .h and .c files, readme file, and design document. Finally try to organize your files under different directories based on their types.

Your design document should be called design.pdf and it should be in PDF format and should be in the same directory. Formats other than PDF are not acceptable; please convert other formats (Word, LaTeX, HTML, …) to PDF. Your design should describe the design of your assignment in enough detail that a knowledgeable programmer could duplicate your work. This includes descriptions of the data structures you use, all non-trivial algorithms and formulas, and a description of each function including its purpose, inputs, outputs, and assumptions it makes about the inputs or outputs.

Deliverables Summary:

1. Design Document (.pdf)
2. README.txt
3. Patch files.

Grade

This assignment is worth 10% of the overall course grade. The assignment will be graded on a 100% grade scale, and then will be scaled down to the 10% its worth. The grading of the assignment will be broken down as follows:

4. 10 % for just submitting a meaningful assignment before or on the due date. This 10% does not account for the full correctness of your assignment but submitting an empty assignment without code will definitely results in loosing this 10% and consequently the whole grade of this assignment.
5. 50 % for the correctness of the functionality and the quality of your code.
6. 15 % for the quality of your inline documentation and the readme file.
7. 25 % for the design document.

Bonus

You can get a bonus worth 10% of the assignment grade if you grab the source code of the ***ps*** program and modified it such that it prints information about the pfork, specifically: pfork_active_id, pfork_standby_id, and pfork_status.

Delays

You have up to 2 working days of delay, after which the assignment will not be accepted and your grade in that case will be ZERO. For every day (of the 2 allowed days), a penalty of 10% will be deducted from the grade. And of course you will lose the 10% mentioned in point 1 above under the "Grade" section.

Important Note

One of the main objectives of this assignment, as stated above, is to learn how to hack the kernel code, and this requires you to read the kernel code before modification. To set the right expectations, getting help in how to navigate the kernel code and understand the needed parts for this assignment will defeat the whole purpose of the assignment. Consequently you should expect less help from the TA regarding that. Simply you need to read the code and try your best to understand it before coming for questions. We will be happy to help you with your questions, but you need to show that you did considerable amount of effort.

Sample User Space Program

```
#include <pfork.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>


int main ( int argc, char ** argv )
{
	pid_t cpid = pfork();
	if (cpid !=0 )
		for (;wait(NULL) > 0 ;);
	else
	{
		char who[10];
		memset (who,10,0);
		if ( pfork_who() == 2 )
			strcpy (who,"STANDBY");
		else strcpy (who,"ACTIVE");
		for ( int  i = 0 ; i  < 5 ; i ++ )
		{
			long int status = get_pfork_status();
			if ( status == i )
			{
				printf ("%s: Current Status is %d\n",who,status);
				sleep ((i+1)*10);
				set_prork_status(i+1);
				printf ("%s: Set Status is %d\n",who,status+1);
			}
		}
	}
	return 0;
}
```