Homework 1

CSE 4600 (Section 01) – Operating Systems - Spring 2022

Submitted to
Department of Computer Science and Engineering
California State University, San Bernardino, California

by:
Spencer Wallace (007463307)

Date: March 4th, 2022

Email:
007463307@coyote.csusb.edu

*Github repository containing all programs

https://github.com/SpencerDWallace/CSE4600/tree/master/Spencer-Wallace-007463307-Homework1

Part 1

1. Process creation via fork()

Question: How many processes does the following piece of code create? Why?

```
Int main(){
fork();
fork();
fork();
return 0;
}
```

Answer: The above piece of code will create a total of 8 processes, including the parent process. I believe 2^n can generally be used to describe the total number of processes resulting n fork calls. This is because the main parent process will call fork, resulting in 2 total process (1*2), each of these will then call the next fork resulting in 4 (2*2), and again each of these will call fork resulting in 8 total (4*2), or 2^3.

Question: Write a C/C++-program that creates a chain of 10 processes and prints out their process ids and relationships. For example, process 1 is the parent of process 2, process 2 is the parent of process 3, process 3 is the parent of 4 and so on. Each child must print out all her ancestors identified by the process IDs

Answer: The C++ program can be found under the github repository linked on the above. The program is titled "fork.cpp" and is under the directory title "part1" . For this program I used a stack that pushed the process ID of the parent process each time a new thread was created. When the new thread is created it prints its ancestor list (goes through the stack), it then creates a new thread and waits until the new thread returns.

-output

```
spencer@spencer-VirtualBox:~/github/CSE4600/Spencer-Wallace-007463307-Homework1/part1$
./fork
Main process has ID: 2449 and has created child with ID: 2450

Process with ID: 2450 has ancestors with IDs: 2449
Process with ID: 2451 has ancestors with IDs: 2450, 2449
Process with ID: 2452 has ancestors with IDs: 2451, 2450, 2449
Process with ID: 2453 has ancestors with IDs: 2452, 2451, 2450, 2449
Process with ID: 2454 has ancestors with IDs: 2453, 2452, 2451, 2450, 2449
Process with ID: 2455 has ancestors with IDs: 2454, 2453, 2452, 2451, 2450, 2449
Process with ID: 2456 has ancestors with IDs: 2455, 2454, 2453, 2452, 2451, 2450, 2449
Process with ID: 2457 has ancestors with IDs: 2456, 2455, 2454, 2453, 2452, 2451, 2450, 2449
Process with ID: 2458 has ancestors with IDs: 2457, 2456, 2455, 2454, 2453, 2452, 2451, 2450, 2449
spencer@spencer-VirtualBox:~/github/CSE4600/Spencer-Wallace-007463307-Homework1/part1$
```

2. Replacing a Process Image

Question: Modify test_exec so that the function excel is used instead of using excelp.

Answer: The modified program is titled “test_exec.cpp” and can be found under the directory “part1”. This program takes the name of an executable as a command-line argument and runs this executable if it is found. For this program I checked to see if there are 2 command line arguments given, where the second argument should be the name of the executable, and use excel to attempt to run the executable.

3. Duplicating a Process Image

Question: Try the test_fork.cpp program and explain what you see on the screen.

Answer When running the program I see the parent and child process output “this is the parent” and “this is the child”, 3 times for the parent and 5 times for the child. Shortly after the parent’s 3rd output the program appears to terminate, and then the child continues to print it’s statement until it has reached 5 prints.

4. Waiting for a Process

Question: Run the program and explain what you have seen on the screen. Modify the program so that the child process creates another child and waits for it. The grandchild prints out the IDs of itself, its parent, and it’s grandparent.

Answer: When running this program I see similar results to the program in the previous section, however now the parent process waits to exit until he child has finished printing its statements. The modified program is titled “test_wait.cpp” and can be found under the directory “part1.” . For this program I used a linked list to create a connection between each thread created and their parent. When the final thread has finished (pid == 0) the list is printed from newest process to oldest process. To make the parent thread wait for its child thread I used the wait call from wait.h

5. Signals

Question: Now try the following scripts (test_signal.cpp). Run the program and hit ^C for a few times. What do you see? Why?

Answer: When running the program I see that, when not pressing ^C, it appears a loop with a print statement is running, where following each print there is a sleep command (which of course I know is true because I can see the code). When pressing ^C it appears the sleep command is interrupting causing the next print statement to run. I would assume this is because ^C sends signal interrupt, which interrupts or terminates the sleep command – but is caught by the statement: (void) signal (SIGINT, func); and allows the program to continue running.

Question: Modify test_signal.cpp by using sigaction() to intercept SIGINT, replace the for loop with while(1); You should be able to quit the program by entering “ ^\ “.

Answer: The modified program is titled “test_sigaction.cpp” and can be found under the directory “part1” . For this program I needed to reference IBM documents on sigaction and sigempty set. Sigaction intercepts is SIGQUIT and quits the program (core dump).

Part 2

1. Pipes – 2. Process Pipes

Question: What do you see when you execute “pipe1” Why? Modify the program pipe1.cpp to pipe1a.cpp so that it accepts a command)=(e.g. “ls -l”) from the keyboard. For example, when you execute “./pipe1a ps -auxw”, it should give you the same output as pipe1.cpp .

Answer: When I execute pipe1 I see the output from a program/command that the pipe ran, in this case it is hard coded that the program/command is “ps -auxw” . The modified program is titled “pipe1a.cpp” and can be found in the directory “part2” . For the program I used the command line arguments to create a c-string which is passed to popen

3. The pipe Call

Question: What do you see when you execute “pipe3” ? Why?

Answer: When I run pipe3 I see one message saying that 5 bytes were sent, and another message saying that 5 bytes were read with an output of what was sent. This is because the message is sent to the pipe with a size of 5bytes (5 chars) and this message is then read from the pipe.

4. Parent and Child Process

Question: Modify pipe4.cpp so that it accepts a message from the keyboard and sends it to pipe5.cpp

Answer: The modified program is titled “pipe4” and can be found in the directory titled “part2” . For this program I again used the command line arguments to create a c-string which is then sent to the pipe.

Part 3

1. Pthreads

Question: Try pthreads.cpp . Modify it so that they run 3 threads (instead of two) and each thread runs a different function, displaying a different message. Copy-and-paste the source code and the outputs in your report.

Answer:

- output

```
spencer@spencer-VirtualBox:~/github/CSE4600/Spencer-Wallace-007463307-Homework1/part3$
./pthread_demo
This is thread function three, my thread name is: Thread 3.
This is thread function one, my thread name is: Thread 1.
This is thread function two, my thread name is: Thread 2.
```

- code

```
spencer@spencer-VirtualBox:~/github/CSE4600/Spencer-Wallace-007463307-Homework1/part3$ cat
pthread_demo.cpp
/***************************************
Student: Spencer Wallace
Instructor: Dr. Khan
CSUSB - CSE 4600

Template provided by Dr Khan
***************************************/
/*
pthreads_demo.cpp
  A very simple example demonstrating the usage of pthreads.
  Compile: g++ -o pthreads_demo pthreads_demo.cpp -lpthread
  Execute: ./pthreads_demo
*/

#include <pthread.h>
#include <stdio.h>
using namespace std;
//The thread
void * thread_func_one (void *data)
{
  char *tname = (char *) data;
  printf("This is thread function one, my thread name is: %s.\n", tname);
  pthread_exit (0);
}

void * thread_func_two (void *data)
{
  char *tname = (char *) data;
  printf("This is thread function two, my thread name is: %s.\n", tname);
  pthread_exit (0);
}
```

```
void * thread_func_three (void *data)
{
 char *tname = (char *) data;
 printf("This is thread function three, my thread name is: %s.\n", tname);
 pthread_exit (0);
}

int main ()
{
 pthread_t id1, id2, id3;          //thread identifiers
 pthread_attr_t attr1, attr2, attr3;  //set of thread attributes
 char *tnames[3] = { "Thread 1", "Thread 2", "Thread 3" }; //names of threads

 pthread_attr_init (&attr1);
 pthread_attr_init (&attr2);
 pthread_attr_init (&attr3);
 //create the threads
 pthread_create (&id1, &attr1, thread_func_one, tnames[0]);
 pthread_create (&id2, &attr2, thread_func_two, tnames[1]);
 pthread_create (&id3, &attr3, thread_func_three, tnames[2]);
 //wait for the threads to exit
 pthread_join (id1, NULL);
 pthread_join (id2, NULL);
 pthread_join (id3, NULL);
 return 0;
}
```

2. Synchronization using Pthreads mutex

Question: Compile and run the above program (shared_resource_mutex). Execute it 5 times in the command prompt. What do you see in the result?

Answer: When running the program I see an output of what seems to be a random-non-repeating number.

Question: Try executing the command with $ time ./{program} . What do you see other than the printf statement? Copy and paste, and report how long the threads took to run.

Answer
-output

```
spencer@spencer-VirtualBox:~/github/CSE4600/Spencer-Wallace-007463307-Homework1/part3$ time
./shared_resource_nomutex
Shared resource value: -32573487

real	0m1.665s
user	0m3.137s
sys	0m0.024s
```

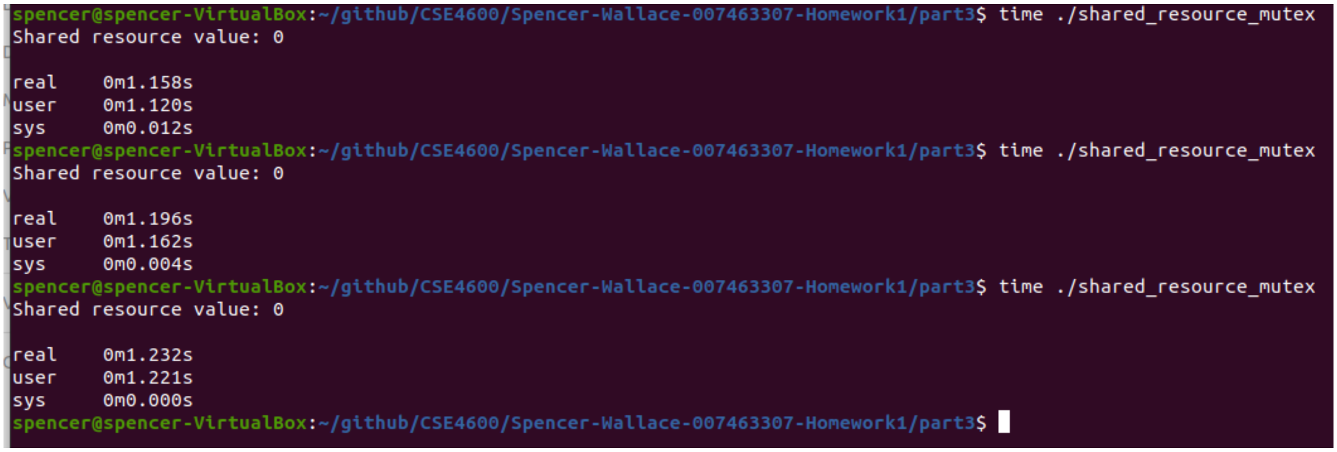
Besides the print statement I see a report of the runtime for the program. I can see from the output that the threads took a total of ~3.137 seconds to run the program (user time).

Question: Try switching the joins
pthread_join(tid2, NULL);
pthread_join(tid1, NULL);
Do you see any difference? Please report if and why you do or do not see any difference in terms of the randomness in the result of the shared resource.

Answer: When switching the order of the joins I do not notice any difference in the output. This is because join does not prevent the threads from performing any operations, it only causes the program to wait for the threads to finish before it continues past the join calls. Since join does not block the threads in any way switching the order creates no difference.

Question: In the inc_dec_resource() function, implement mutual exclusion (pthread_mutex_lock) to ensure that the result becomes 0 every time when you execute your program. Put your updated code in the report (highlighted) and show your screenshot of the execution by running the script three time using $ time ./shared_resource_mutex. Hint: Your loop is incrementing/decrementing the resource which should be protected by each thread while it is executing that portion.

Answer:
-output



```
spencer@spencer-VirtualBox:~/github/CSE4600/Spencer-Wallace-007463307-Homework1/part3$ time ./shared_resource_mutex
Shared resource value: 0

real    0m1.158s
user    0m1.120s
sys     0m0.012s
spencer@spencer-VirtualBox:~/github/CSE4600/Spencer-Wallace-007463307-Homework1/part3$ time ./shared_resource_mutex
Shared resource value: 0

real    0m1.196s
user    0m1.162s
sys     0m0.004s
spencer@spencer-VirtualBox:~/github/CSE4600/Spencer-Wallace-007463307-Homework1/part3$ time ./shared_resource_mutex
Shared resource value: 0

real    0m1.232s
user    0m1.221s
sys     0m0.000s
spencer@spencer-VirtualBox:~/github/CSE4600/Spencer-Wallace-007463307-Homework1/part3$ 
```

-code begins on next page
The program can also be found on github under the directory “part3” with the program titled “shared_resource_mutex.cpp”

```
spencer@spencer-VirtualBox:~/github/CSE4600/Spencer-Wallace-007463307-Homework1/part3$ cat
shared_resource_mutex.cpp
/****************************************
Student: Spencer Wallace
Instructor: Dr. Khan
CSUSB - CSE 4600

Template provided by Dr Khan
****************************************/
/*
Compile: gcc -o shared_resource_mutex shared_resource_mutex.c -lpthread
Execute: ./shared_resource_mutex
*/
#include <stdio.h>
#include <pthread.h>
#define iterations 300000000
long long shared_resource = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
// Thread function to modify shared resource
void* inc_dec_resource(void* arg){
  //get the pointer from main thread and dereference it to put the value in resource_value
  int resource_value = *(int *) arg;
  pthread_mutex_lock(&mutex);
  for(int i=0; i < iterations; i++){
   shared_resource += resource_value;
  }
  pthread_mutex_unlock(&mutex);
  pthread_exit(NULL);
}
int main(void){
  // Thread 1 to increment shared resource
  pthread_t tid1, tid2;
  int value1 = 1;
  pthread_create(&tid1, NULL, inc_dec_resource, &value1);
  // Thread 2 to increment shared resource
  int value2 = -1;
  pthread_create(&tid2, NULL, inc_dec_resource, &value2);
  pthread_join(tid2, NULL);
  pthread_join(tid1, NULL);
  printf("Shared resource value: %lld\n", shared_resource);
  return 0;
}
```