PA-5

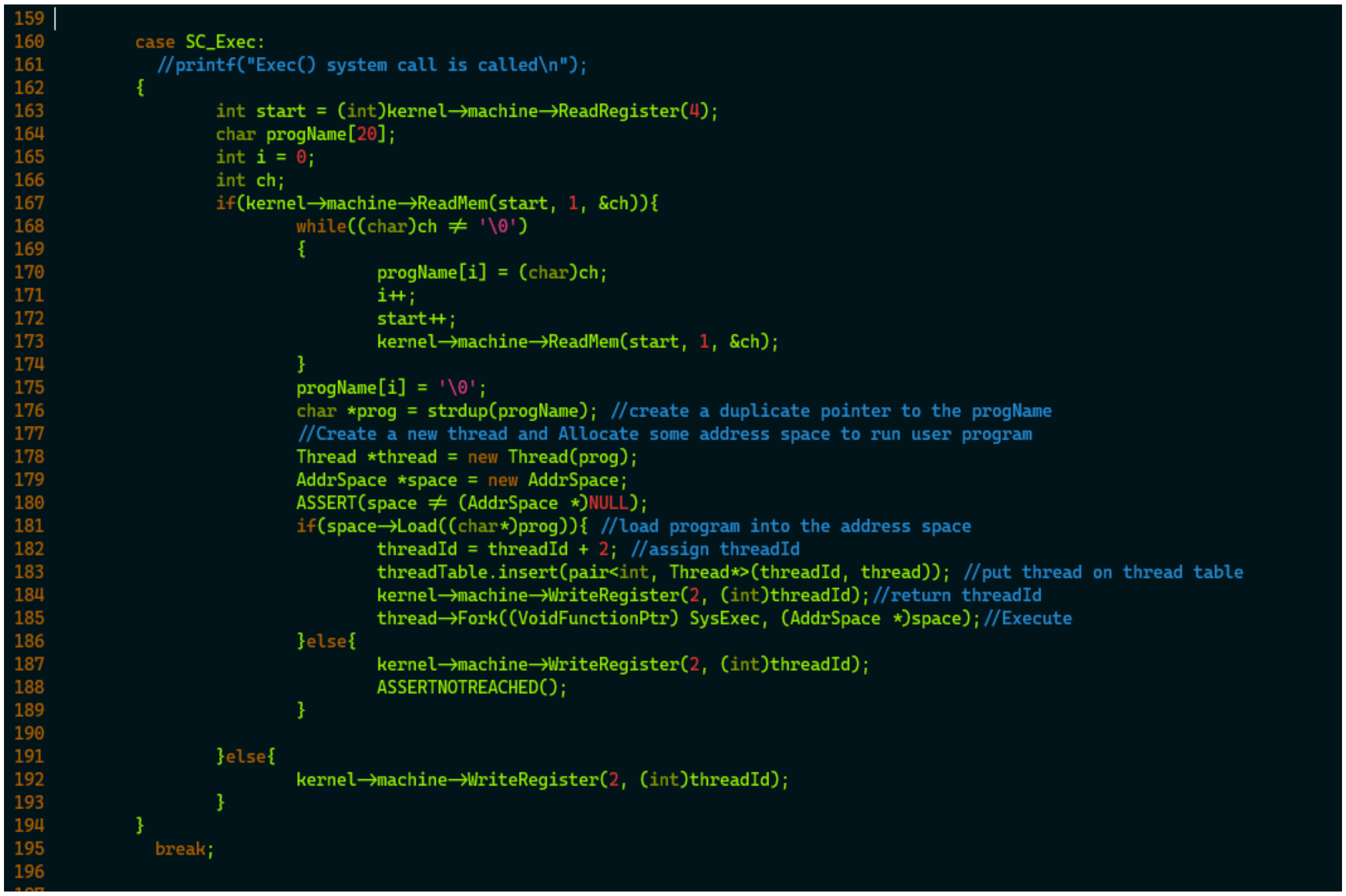
Name: Raghunandan Gajanan Bhat

SUNetID: rgbhat@syr.edu

Task-1: System calls for multi-programming

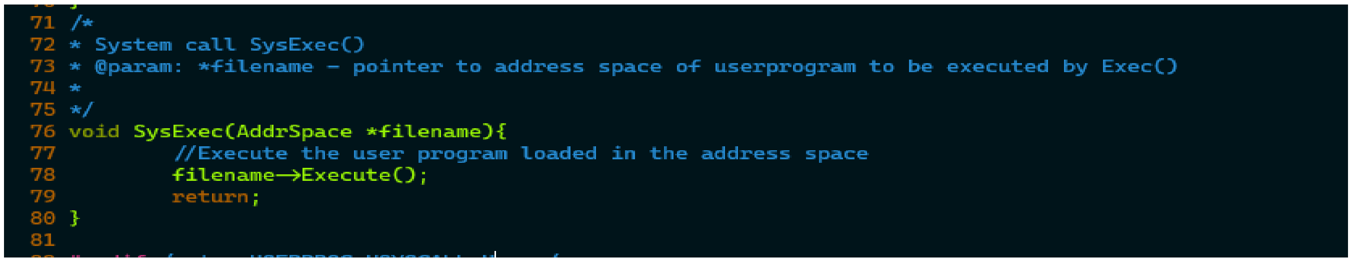
✪ [R-1]:

`Exec()` system call: Reads the filename of user program, creates a new address space and thread for its execution.



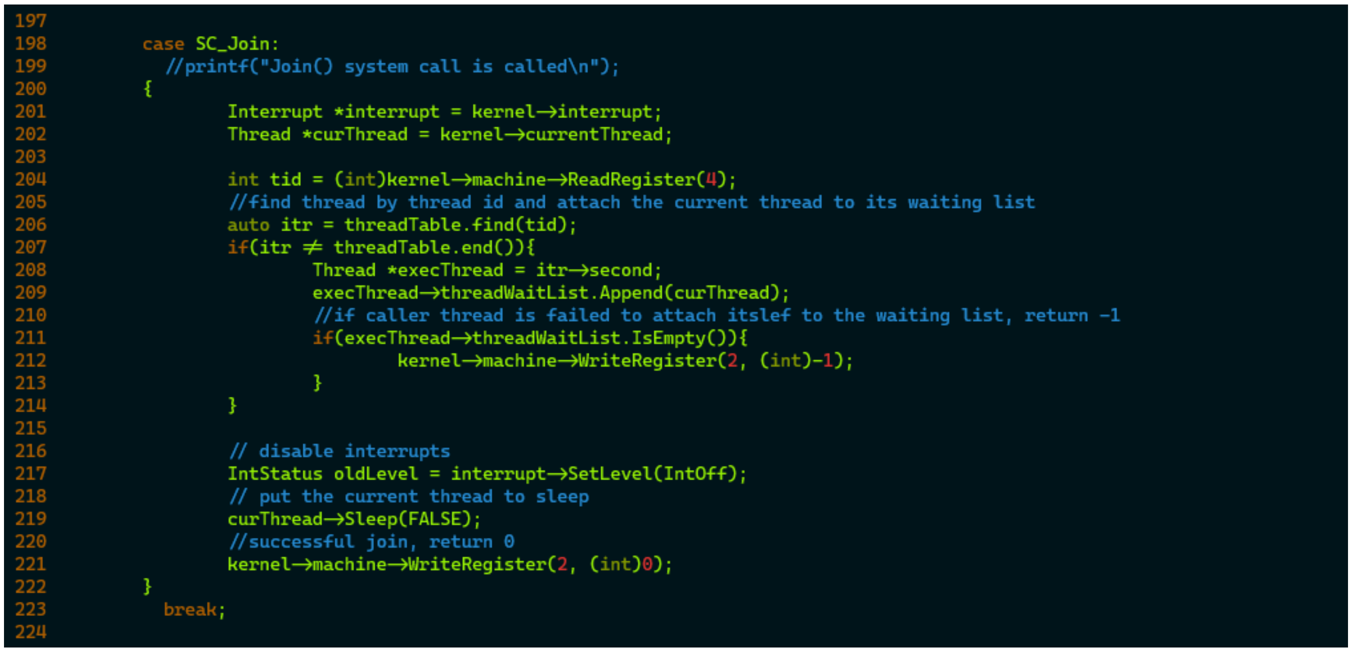
```
        case SC_Exec:
          //printf("Exec() system call is called\n");
        {
                int start = (int)kernel->machine->ReadRegister(4);
                char progName[20];
                int i = 0;
                int ch;
                if(kernel->machine->ReadMem(start, 1, &ch)){
                        while((char)ch != '\0')
                        {
                                progName[i] = (char)ch;
                                i++;
                                start++;
                                kernel->machine->ReadMem(start, 1, &ch);
                        }
                        progName[i] = '\0';
                        char *prog = strdup(progName); //create a duplicate pointer to the progName
                        //Create a new thread and Allocate some address space to run user program
                        Thread *thread = new Thread(prog);
                        AddrSpace *space = new AddrSpace;
                        ASSERT(space != (AddrSpace *)NULL);
                        if(space->Load((char*)prog)){ //load program into the address space
                                threadId = threadId + 2; //assign threadId
                                threadTable.insert(pair<int, Thread*>(threadId, thread)); //put thread on thread table
                                kernel->machine->WriteRegister(2, (int)threadId);//return threadId
                                thread->Fork((VoidFunctionPtr) SysExec, (AddrSpace *)space);//Execute
                        }else{
                                kernel->machine->WriteRegister(2, (int)threadId);
                                ASSERTNOTREACHED();
                        }

                }else{
                        kernel->machine->WriteRegister(2, (int)threadId);
                }
        }
          break;
```

`SysExec()` in `userprog/ksyscall.h`



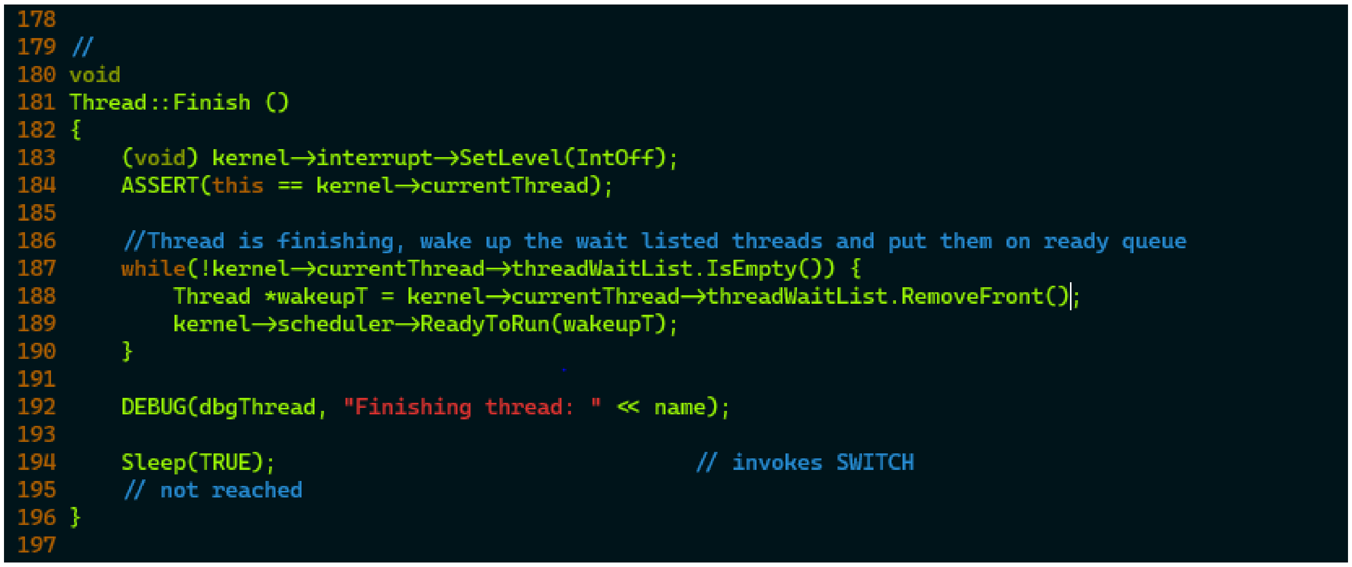
```
/*
* System call SysExec()
* @param: *filename - pointer to address space of userprogram to be executed by Exec()
*
*/
void SysExec(AddrSpace *filename){
        //Execute the user program loaded in the address space
        filename->Execute();
        return;
}
```

Implementation of `Join()`:



```
        case SC_Join:
          //printf("Join() system call is called\n");
        {
                Interrupt *interrupt = kernel->interrupt;
                Thread *curThread = kernel->currentThread;

                int tid = (int)kernel->machine->ReadRegister(4);
                //find thread by thread id and attach the current thread to its waiting list
                auto itr = threadTable.find(tid);
                if(itr != threadTable.end()){
                        Thread *execThread = itr->second;
                        execThread->threadWaitList.Append(curThread);
                        //if caller thread is failed to attach itslef to the waiting list, return -1
                        if(execThread->threadWaitList.IsEmpty()){
                                kernel->machine->WriteRegister(2, (int)-1);
                        }
                }

                // disable interrupts
                IntStatus oldLevel = interrupt->SetLevel(IntOff);
                // put the current thread to sleep
                curThread->Sleep(FALSE);
                //successful join, return 0
                kernel->machine->WriteRegister(2, (int)0);
        }
          break;
```

Wake-up the wait listed threads in `Thread::Finish()`



```
//
void
Thread::Finish ()
{
    (void) kernel->interrupt->SetLevel(IntOff);
    ASSERT(this == kernel->currentThread);

    //Thread is finishing, wake up the wait listed threads and put them on ready queue
    while(!kernel->currentThread->threadWaitList.IsEmpty()) {
        Thread *wakeupT = kernel->currentThread->threadWaitList.RemoveFront();
        kernel->scheduler->ReadyToRun(wakeupT);
    }

    DEBUG(dbgThread, "Finishing thread: " << name);

    Sleep(TRUE);                                // invokes SWITCH
    // not reached
}
```

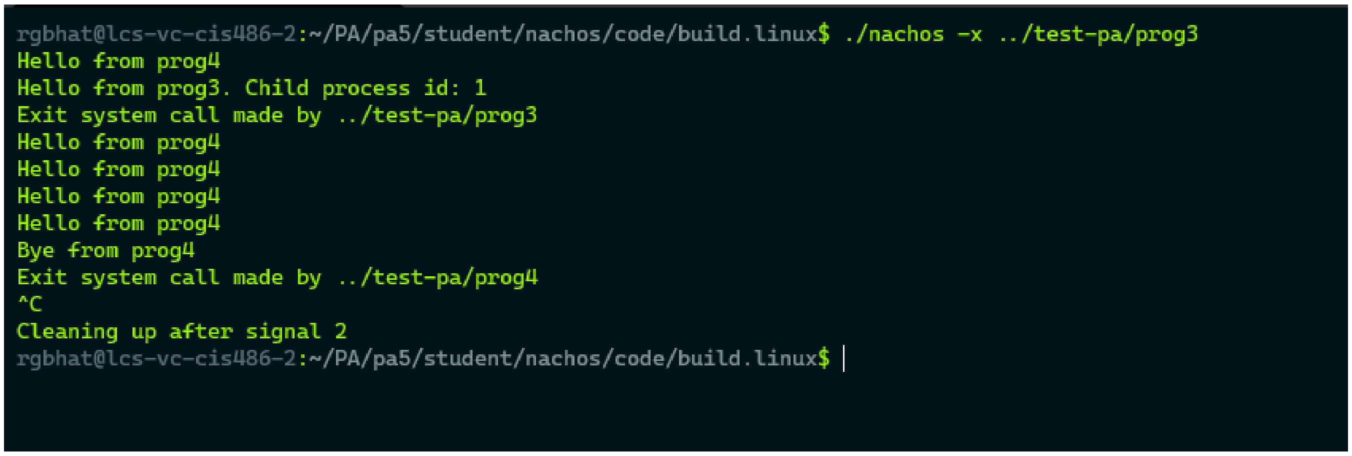
✪ **[R-2]:**

1. User programs
   prog3.c : The test program `prog3.c` calls `prog4.c` using `Exec()` system call and `Exec()` returns a `Space Id` which is a unique thread ID. The function `strcpy()` copies the message "Hello from prog3. Child process id: " message to `str` variable. Then we attach the `Space Id` returned by the `Exec()` system call to `str`

variable using `itoa()`. Using the `Write()` system call, the message is written to the output and Exit() system call is used to finish execution.

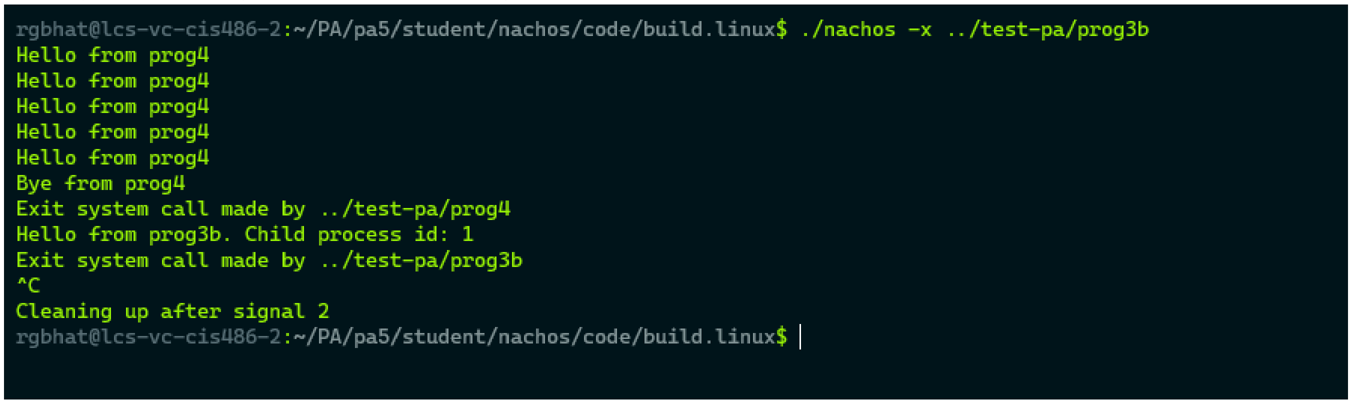
prog3b.c: The test program `prog3.c` calls `prog4.c` using `Exec()` system call and `Exec()` returns a `Space Id` which is a unique thread ID. Then prog3b calls Join() system call which puts prog3b into sleep until prog4 is finished. If the prog3b waits for prog4 Join() returns 0 and prog3b prints the message "Hello from prog3b. Child process id: " along with the Space Id of prog4. In case if Join() fails, prog3b prints "Failed to Join". The messages are written to output using Write() system call and Exit() is called to finish execution.
prog4.c: This program declares two messages - "Hello from prog4" and "Bye from prog4". Then using the Write() system call "Hello from prog4" is written to output 5 times and "Bye from prog4" is written once. Then prog4 exits using Exit() system call.

2. `./nachos -x ../test-pa/prog3`
   When prog3 starts executing, it first calls Exec() and executes prog4. The prog4 prints the first line and prog3 continues its execution and finishes first. Here prog3 does not wait for prog4. Then prog4 continues execution and prints the hello messages and exits.



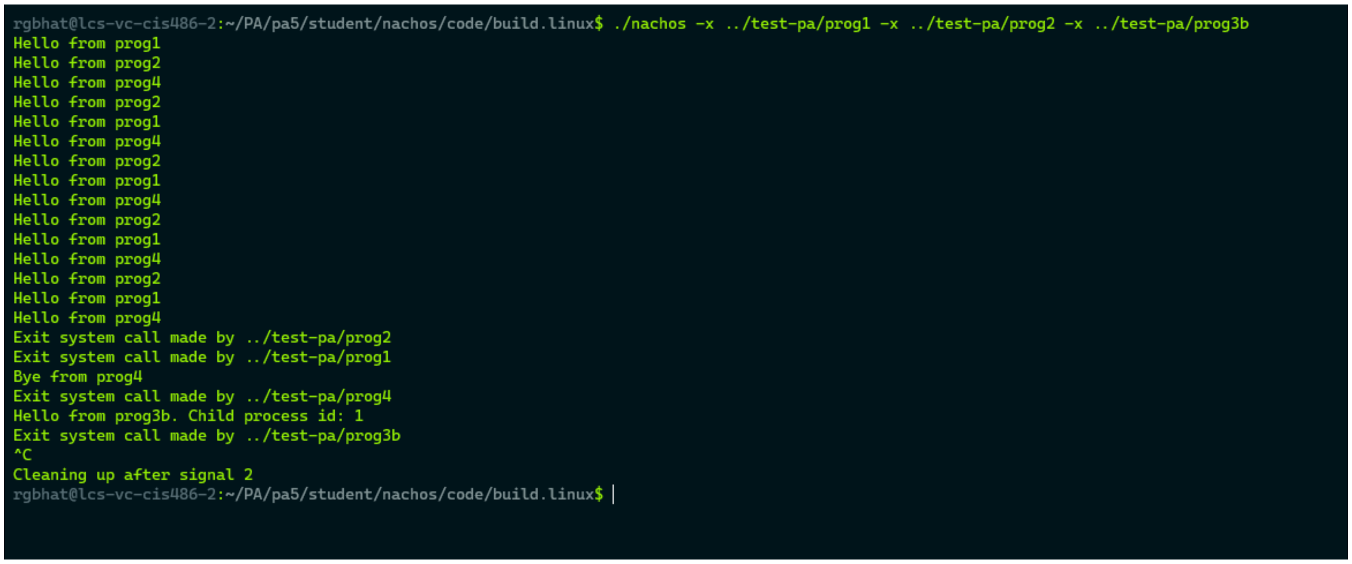
```
rgbhat@lcs-vc-cis486-2:~/PA/pa5/student/nachos/code/build.linux$ ./nachos -x ../test-pa/prog3
Hello from prog4
Hello from prog3. Child process id: 1
Exit system call made by ../test-pa/prog3
Hello from prog4
Hello from prog4
Hello from prog4
Hello from prog4
Bye from prog4
Exit system call made by ../test-pa/prog4
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa5/student/nachos/code/build.linux$
```

3. `./nachos -x ../test-pa/prog3b`
   prog3b calls prog4 using the Exec() system call and using the Space Id returned by the Exec(), prog3b waits for prog4 to complete its execution. Join() system call puts prog3b into sleep and executes the prog4 and prints the messages. Then prog4 calls Exit() and prog3b resumes execution and exits.



```
rgbhat@lcs-vc-cis486-2:~/PA/pa5/student/nachos/code/build.linux$ ./nachos -x ../test-pa/prog3b
Hello from prog4
Hello from prog4
Hello from prog4
Hello from prog4
Hello from prog4
Bye from prog4
Exit system call made by ../test-pa/prog4
Hello from prog3b. Child process id: 1
Exit system call made by ../test-pa/prog3b
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa5/student/nachos/code/build.linux$
```

4. `./nachos -x ../test-pa/prog1 -x ../test-pa/prog2 -x ../test-pa/prog3b`
   Nachos executes all programs one by one. Each user program gets to use the CPU for certain amount of time until it is interrupted. When interruption happens, the CPU switches to a different thread in the ready queue. When prog3b calls prog4, prog3b goes to sleep by calling Join() system call. Then prog3b waits for prog4 to finish. But in the ready queue prog1 and prog2 still exists and after each interrupt threads are switched. Therefore, each of prog1, prog2 and prog4's messages are printed one after other. Once prog4 is finished, it wakes up all the threads in its wait list(prog3b). prog3b resumes execution, prints the message and exits.



```
rgbhat@lcs-vc-cis486-2:~/PA/pa5/student/nachos/code/build.linux$ ./nachos -x ../test-pa/prog1 -x ../test-pa/prog2 -x ../test-pa/prog3b
Hello from prog1
Hello from prog2
Hello from prog4
Hello from prog2
Hello from prog1
Hello from prog4
Hello from prog2
Hello from prog1
Hello from prog4
Hello from prog2
Hello from prog1
Hello from prog4
Hello from prog2
Hello from prog1
Hello from prog4
Exit system call made by ../test-pa/prog2
Exit system call made by ../test-pa/prog1
Bye from prog4
Exit system call made by ../test-pa/prog4
Hello from prog3b. Child process id: 1
Exit system call made by ../test-pa/prog3b
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa5/student/nachos/code/build.linux$ 
```