CMPSC 311 SCRIBE NOTES

Lecture 19 - Threads

Xiaoyan Wu
Dan Witman
Christopher Whitridge
Rodney Wells

Overview

Each process of a multiprocess program holds separate code, data, heap, and stack in virtual memory. The redundancy (e.g., libraries in code) may cause significant memory wastage and slows down the program. An alternative to processes is to use threads which share code, data, and heap but maintain their own registers and stacks.

Agenda

- Multi-threaded Programming
- P-Thread

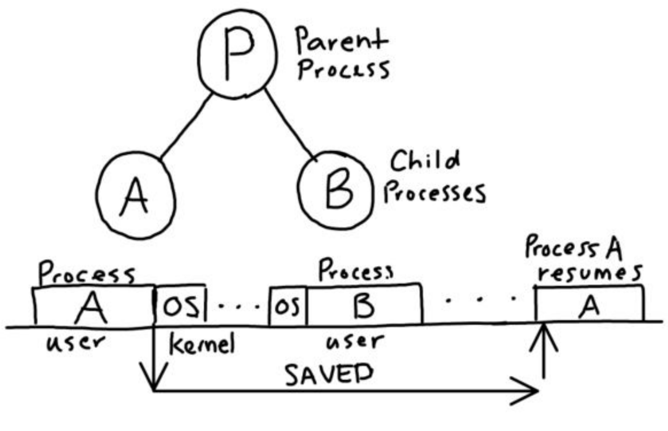
Multi-Thread

- What is a thread?
  - A thread is an abstraction of a "process activity". We define a process activity as simply an active sequence of instructions within a process
  - Threads exist *within* a process, and thus multiple threads can be running inside of a single process
- Threads share the following:
  - Code
  - Data
  - Heap
- Thread maintain their own:
  - Stack
  - Registers (include program counter)

Thread vs. Process

Both processes and threads are independent sequences of execution. Threads exist in a process, and thereby share same address space. Processes, on the other hand, are instances of executing program.

This diagram below illustrates how sharing of a CPU among multiple processes (or threads)



(*above*) Figure 1. Multiple processes

This diagram demonstrates how the OS handles changes between processes. It must save where process "A" was interrupted so that it remembers the instruction upon resuming the process later. Similarly, it must remember where process “B” ended so that it can resume later.

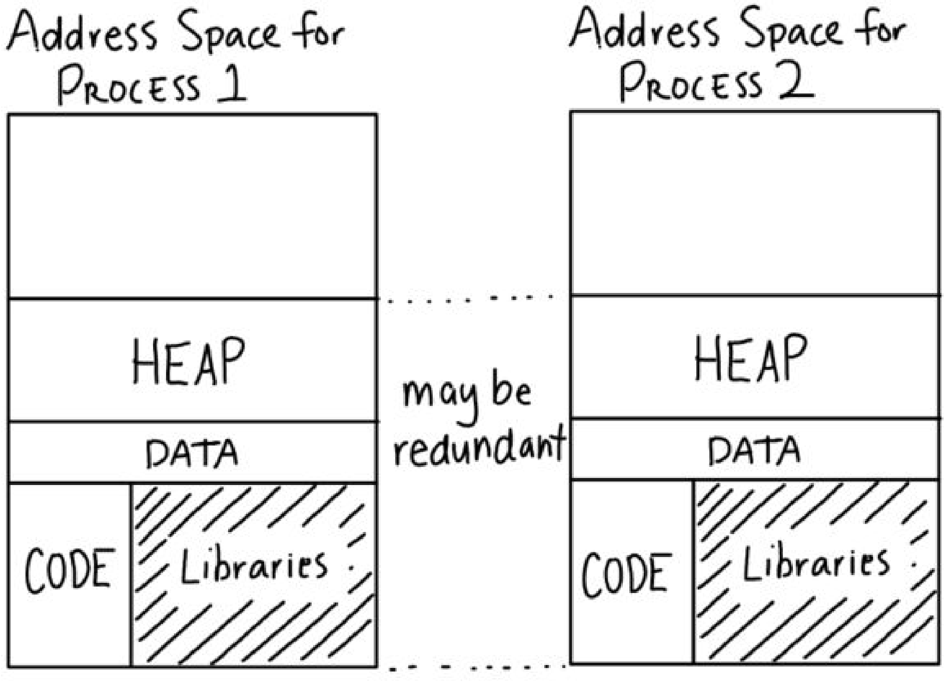
Multiprogramming

- The OS keeps track of several processes in memory at a time (as seen in Figure 1 above), and can execute any of them.
- The address spaces between processes are disjointed, meaning they do not share stacks, heaps, code, data, libraries, etc. A visual example of these separated address spaces is in Figure 2a.

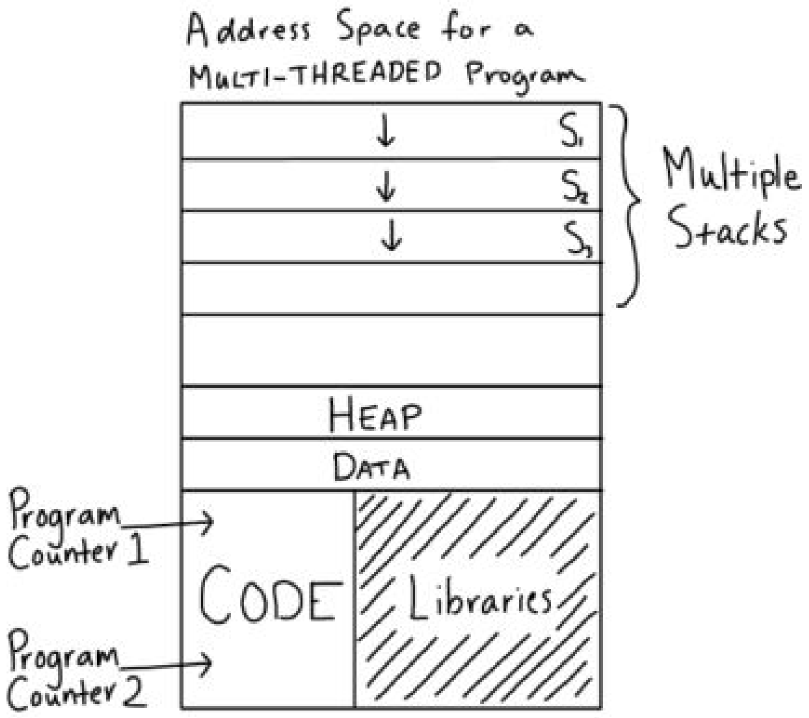
Multithreading

- A single process can host multiple threads of instructions at a time
- These threads all share the same address space. This is very helpful because now the threads have access to the same heap, data, code, and libraries (Figure 2b). However, the threads *do not* share the same stack or program counter. Separate stacks are maintained because different threads will have a different set of function calls. To maintain the information about these function calls and the information passed between them, each thread requires its own stack.

The diagrams below show the difference between multiprogramming and multithreading:



(*above*) Figure 2a. *Multiprogramming* - Two different address spaces must be created to maintain two different processes. Note how libraries and the code base are duplicated, wasting memory.



(*above*) Figure 2b. *Multithreading* - Only one address is needed to maintain a multi-threaded process.

Pthread:

In POSIX standard, we are using the libraries of pthread, where pthread_t is the type of the identifier. Similar to process, <pthread.h> also provides a set of function to manipulate pthreads.

Equivalents between Processes and Pthreads

processes	p threads
`getpid()`	`pthread_t pthread_self()`
`if (childpid == …)`	`int pthread_equal(pthread_t,pthread_t)`

`fork(), exec()`	`int pthread_create(...)`

`pthread_self () -> pthread_t`

- This function takes no (void) parameter and returns the calling thread identifier.
- This function always succeeds hence no error will occur.

`pthread_equal (pthread_t, pthread_t) -> int`

- This function takes two thread identifiers and compares them with each other. If two threads are equal then a non-zero value is returned, otherwise 0 is returned.
- This function always succeeds hence no error will occur.

`pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)`

- This function starts a new thread in calling process and start execution by invoking start_routine(). *args is the argument of start_routine().
- The new thread is terminated if any of the following happens:
    1. pthread_exit() is called with exit status.
    2. start_routine() returns
    3. pthread_cancel() is called
    4. main thread returns from main(), then all threads will be terminated
- Argument *attr is pointing to a pthread_attr_t struct which is used to give attributes to the thread that is created. A pthread_attr_t object is initialized by pthread_attr_init(). If *attr is NULL, then default attributes will be used.
- If pthread_create succeeds, the ID of the new thread will be stored in buffer and pointed by *thread, then 0 will be returned.
- If pthread_create failed, the following are possible errors:
    1. EAGAIN: there is no enough resource to create a new thread or the number of thread reach the system limit.
    2. EINVAL: *attr is invalid.
    3. EPERM: no permission to set the scheduling policy and parameters specified in attr.

Project 2

We discussed some of the concepts behind the new implementation of wordcount in project 2. Mainly, project 2 will have the same functionality as project 1; it will perform a word count operation on an input file. However, in this implementation, we will use multi-process programming to count the words. Call this new implementation wordc-mp (for wordc-multi-process). The below diagram visualizes generally how the project will look:

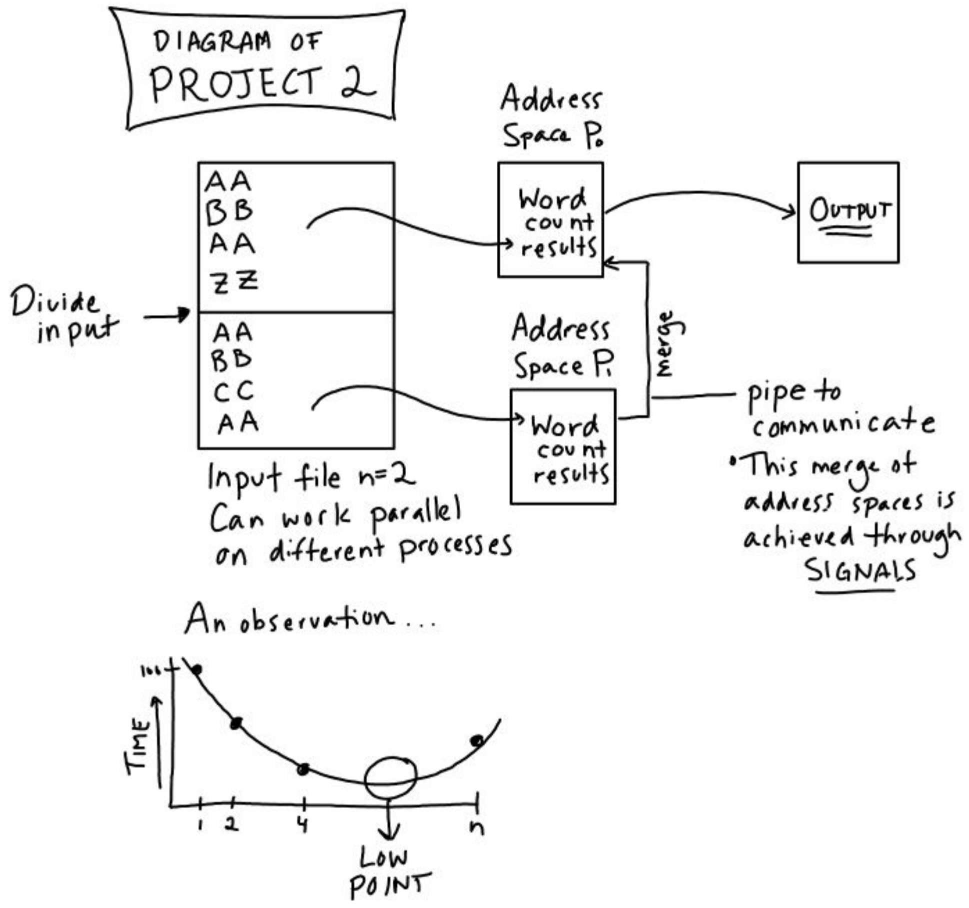


Figure 3. Project 2 Diagram

Note: At the bottom of Figure 3 is a graph of n (number of truncations of an input file that will be allocated its own process) and execution time. We can see that as n increases, there will be a point where it is no longer advantageous to divide the file for multiple processes, and the execution time will actually start to increase.

References:

Linux man page:

1. http://man7.org/linux/man-pages/man3/pthread_self.3.html
2. http://man7.org/linux/man-pages/man3/pthread_attr_init.3.html
3. http://man7.org/linux/man-pages/man3/pthread_create.3.html
4. http://man7.org/linux/man-pages/man3/pthread_equal.3.html

Other resource:

1. http://softpixel.com/~cwright/programming/threads/threads.c.php
2. http://stackoverflow.com/questions/200469/what-is-the-difference-between-a-process-and-a-thread
3. http://www.programmerinterview.com/index.php/operating-systems/thread-vs-process/