

CSC209 Week 8 Notes

Hyungmo Gu

May 17, 2020

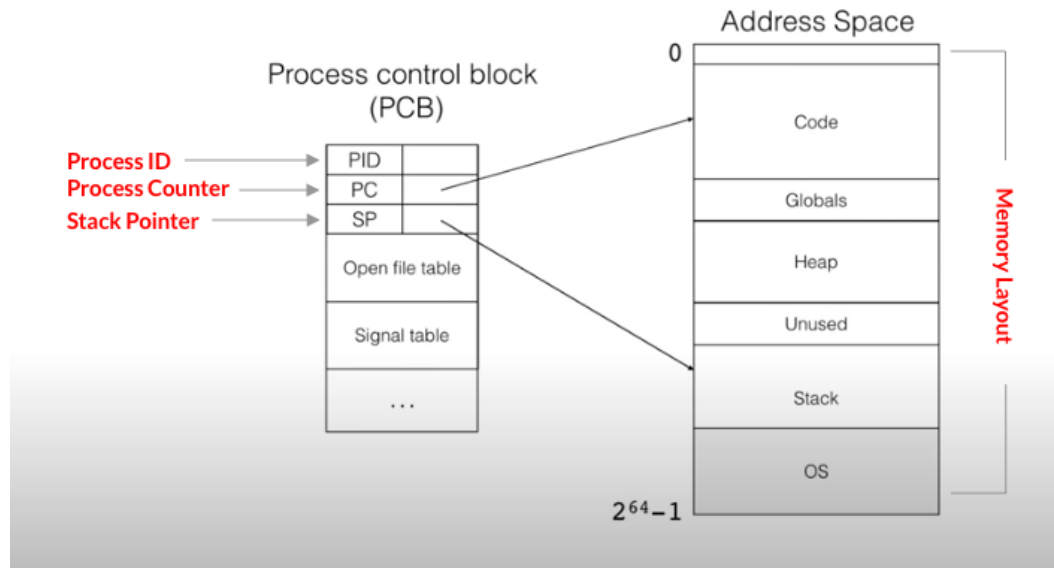
Processes 1 of 8

- Process Models
 - Program
 - * The executable instructions of a program
 - * Source code
 - * Compiled machine code

```
0000000: cffa edfe 0700 0001 0300 0080 0200 0000 .....
0000010: 1000 0000 1005 0000 8500 2000 0000 0000 .....
0000020: 1900 0000 4800 0000 5f5f 5041 4745 5a45 ....H...PAGEEE
0000030: 524f 0000 0000 0000 0000 0000 0000 0000 NO.....
0000040: 0000 0000 0100 0000 0000 0000 0000 0000 .....
0000050: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000060: 0000 0000 0000 0000 1900 0000 2802 0000 .....{...
0000070: 5f5f 5445 5854 0000 0000 0000 0000 0000 ...TEXT.....
0000080: 0000 0000 0100 0000 0010 0000 0000 0000 .....
0000090: 0000 0000 0000 0000 0010 0000 0000 0000 .....
00000a0: 0700 0000 0500 0000 0600 0000 0000 0000 .....
00000b0: 5f5f 7465 7874 0000 0000 0000 0000 0000 ...text.....
00000c0: 5f5f 5445 5854 0000 0000 0000 0000 0000 ...TEXT.....
00000d0: b00d 0000 0100 0000 a701 0000 0000 0000 .....
00000e0: b00d 0000 0400 0000 0000 0000 0000 0000 .....
00000f0: 0004 0080 0000 0000 0000 0000 0000 0000 .....
000100: 5f5f 7374 7562 7300 0000 0000 0000 0000 ...stubs.....
000110: 5f5f 5445 5854 0000 0000 0000 0000 0000 ...TEXT.....
000120: 580f 0000 0100 0000 0c00 0000 0000 0000 X.....
000130: 580f 0000 0100 0000 0000 0000 0000 0000 X.....
000140: 0804 0080 0000 0000 0600 0000 0000 0000 .....
000150: 5f5f 7374 7562 5f68 656e 7065 7200 0000 ...stub_helper...
000160: 5f5f 5445 5854 0000 0000 0000 0000 0000 ...TEXT.....
000170: 640f 0000 0100 0000 2400 0000 0000 0000 d.....$.....
000180: 640f 0000 0200 0000 0000 0000 0000 0000 d.....
000190: 0004 0080 0000 0000 0000 0000 0000 0000 .....
0001a0: 5f5f 6373 7472 496e 4700 0000 0000 0000 ...cstring.....
0001b0: 5f5f 5445 5854 0000 0000 0000 0000 0000 ...TEXT.....
0001c0: 880f 0000 0100 0000 0d00 0000 0000 0000 .....
0001d0: 880f 0000 0000 0000 0000 0000 0000 0000 .....
0001e0: 0200 0000 0000 0000 0000 0000 0000 0000 .....
0001f0: 5f5f 756e 7769 6e64 5f69 6e66 6f00 0000 ...unwind_info...
000200: 5f5f 5445 5854 0000 0000 0000 0000 0000 ...TEXT.....
000210: 980f 0000 0100 0000 4800 0000 0000 0000 .....H.....
000220: 980f 0000 0200 0000 0000 0000 0000 0000 .....
000230: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000240: 5f5f 4548 5f46 7241 6465 0000 0000 0000 ...eh_frame.....
000250: 5f5f 5445 5854 0000 0000 0000 0000 0000 ...TEXT.....
000260: a00f 0000 0100 0000 1800 0000 0000 0000 .....
000270: a00f 0000 0300 0000 0000 0000 0000 0000 .....
000280: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000290: 1900 0000 a800 0000 5f5f 4441 5441 0000 .....DATA.....
0002a0: 0000 0000 0000 0000 0010 0000 0100 0000 .....
0002b0: 0010 0000 0000 0000 0010 0000 0000 0000 .....
0002c0: 0010 0000 0000 0000 0700 0000 0300 0000 .....
```

- Process
 - * Running instance of program
- Process Control Block (PCB)

- * is a data structure used by computer operating system to store all information about process.
- * Code, Global, Heap and Stack together are called the **State of Program**



- **Code:** is program code
 - **Stack:** tells which function is being executed, and hold values of local variables
 - **Heap and Globals:** holds current value for other variables in the program
- * is also known as **process descriptor**
 1. When a process is created (initialized or installed), the operating system creates a corresponding process control block.
 2. Information in a process lblock is updated during the transition of process states
 3. When the process terminates, its PCB is returned to the pool from which new PCBs are drawn
 4. Each process has a single PCB

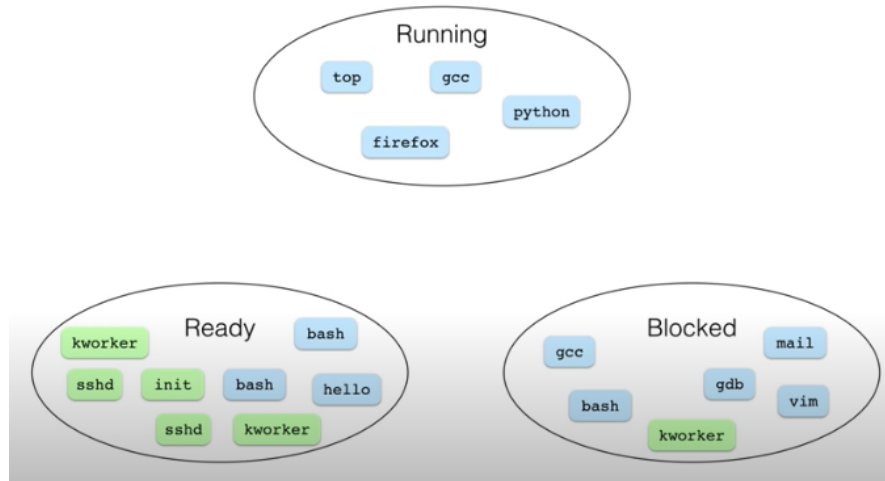
– Top

- * is a basic unix command useful for observing current state of Unix System.

```
top - 14:23:18 up 11 days, 4:50, 13 users, load average: 0.00, 0.05, 0.09
Tasks: 666 total, 1 running, 662 sleeping, 3 stopped, 0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 65954776k total, 46558744k used, 19396032k free, 33357016k buffers
Swap: 102399996k total, 0k used, 102399996k free, 9139880k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8	root	20	0	0	0	0	S	0	0.0	12:03.39	rcu_sched
2196	root	39	19	0	0	0	S	0	0.0	47:40.80	kipmi0
13365	reid	20	0	17732	1768	964	R	0	0.0	0:00.54	top
23412	root	20	0	0	0	0	S	0	0.0	0:21.41	kworker/2:0
51571	c4user	24	4	30668	5960	1228	S	0	0.0	8:47.88	tmux
1	root	20	0	24452	2372	1336	S	0	0.0	0:09.76	init

– Three-state Process Management Model



- * Works with PCB
- * Works simultaneously on multiple processors
- * A program is in either one of three states:
 - **Running:** The process that is currently being executed
 - **Ready:** A process that is queuing and prepare to execute when given the opportunity
 - **Blocked:** A process that cannot execute until some event occurs, i.e. waking up from sleep

Processes 2 of 8

- Creating Processes with Fork

– `fork()`

- * is a part of `unistd.h` library.
- * creates a new process, which is called child process, which runs
- * starts executing after `fork()` is called

```

1  #include <stdio.h>
2  #include <unistd.h> // <- fork imported here
3
4  int main() {
5      int i;
6      pid_t result;
7
8      i = 5;
9      printf("%d\n", i);

```

```

10
11     result = fork(); // <- Here is the fork :)
12
13     if (result > 0) {
14         i = i + 2;
15     } else if (result == 0) {
16         i = i - 2;
17     } else {
18         perror("fork");
19     }
20
21     printf("%d\n", i);
22     return 0;
23 }
24

```

Listing 1: process_example_1.c

Processes 3 of 8

- Process Relationship and Termination (Part 1)
 - Parent and child processes don't occur in order
 - Child processes not in order since they are running concurrently
 - Key Question: How to make the parent to wait before child finishes?

```

1 // Output of process_example_2.c
2 // Run by typing gcc -Wall process_example_2.c
3
4 [20480] Original process (my parent is 15738)
5 [20481] Child 0 0
6 [20481] Child 0 1
7 [20482] Child 1 0 //<- notice 1 starts before 0 ends
8 [20481] Child 0 2
9 [20482] Child 1 1
10 [20481] Child 0 3
11 [20483] Child 2 0
12 [20482] Child 1 2
13 [20481] Child 0 4
14 [20480] Parent about to terminate // <- notice parent ends
    before children
15 [20482] Child 1 3
16 [20483] Child 2 1
17 [20484] Child 3 0 //<- notice 3 starts before 1 and 2 ends
18 [20482] Child 1 4
19 [20483] Child 2 2
20 [20484] Child 3 1
21 [20485] Child 4 0
22 [20483] Child 2 3

```

```

23      [20484] Child 3 2
24      [20483] Child 2 4
25      [20485] Child 4 1
26      [20484] Child 3 3
27      [20485] Child 4 2
28      [20484] Child 3 4
29      [20485] Child 4 3
30      [20485] Child 4 4
31

```

Processes 4 of 8

- wait
 - **Syntax:** `pid_t wait(int *wstatus)`
 - Is a part of *sys/wait.h* library.
 - Blocks the calling process until one of its child processes exists or a signal is received.
 - Parent continues its execution after wait call
- WIFEXITED
 - **Syntax:** `int WIFEXITED (int status)`
 - Checks if child exited normally
 - Returns non-zero if true
- WIFEXITSTATUS(status)
 - **Syntax:** `int WIFEXITSTATUS (int status)`
 - Returns code when child exits
- WTERMSIG
 - **Syntax:** `int WTERMSIG (int status)`
 - Returns non-zero value if the child process terminated due to a signal, i.e. `abort()`

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4  #include <signal.h>
5
6  int main() {
7      ...
8
9      for(i = 0; i < 5; i++) {

```

```

10     pid_t pid;
11     int status;
12     if ((pid = wait(&status)) == -1) { // <- wait called here
13         perror("wait");
14     } else {
15         if (WIFEXITED(status)) { // <- processes ended
normally
16             printf("Child %d terminated with %d\n", pid,
WEXITSTATUS(status));
17         } else if (WIFSIGNALED(status)) { // <- child exited
due to signal
18             printf("Child %d terminated with signal %d\n", pid
, WTERMSIG(status));
19         } else {
20             printf("Shouldn't get here");
21         }
22     }
23 }
24 printf("[%d] Parent about to terminate\n", getpid());
25 return 0;
26 }
27

```

Listing 2: process_example_3.c

Processes 6 of 8

- Running Different Programs

- `execl`

- * **Syntax:** `int execl(const char *filepath, [const char *arg1, const char *arg2], NULL)`
- * is a part of *unistd.h* library
- * Requires *NULL* as final argument
- * Code doesn't return to main if `execl` is successful
- * Code returns to main if `execl` is unsuccessful

```

1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main() {
5      printf("About to call execl. My PID is %d\n", getpid());
6      execl("./hello.out", NULL); // <- searches for .out
7      perror("exec");
8      return 1;
9  }
10

```

Listing 3: process_example_4.c