



Department of “Computer Engineering”

“Operating Systems”

Project Interim Report

Lecturer: Prof. Dr. Ercan SOLAK

Leyla Abdullayeva - 1904010038

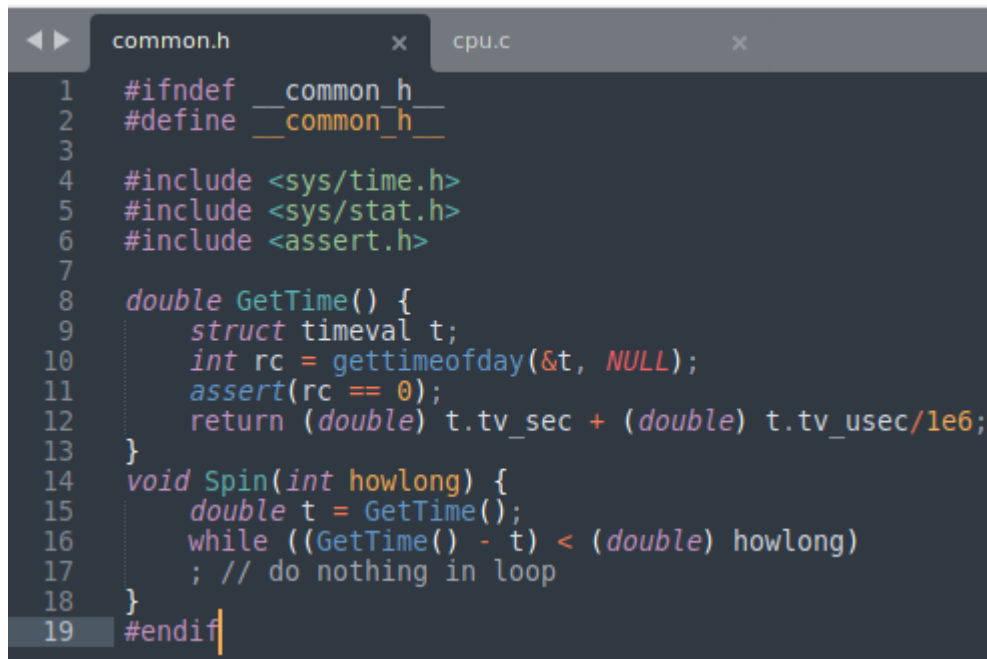
Darina Karimova - 2104011002

Part 1

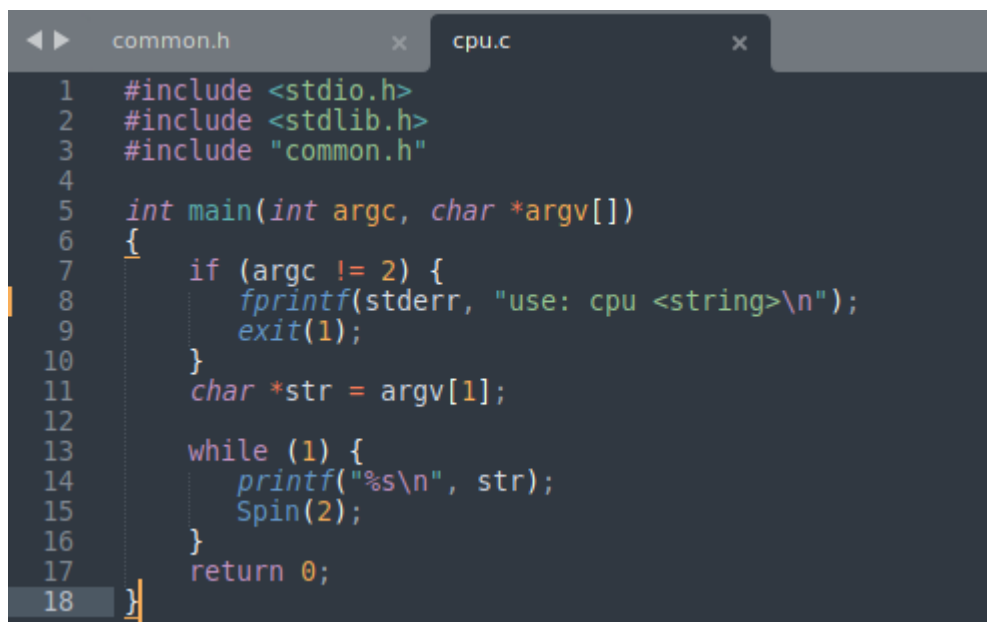
- Virtualization

One of the fundamental functions of Operating Systems is Virtualization. Through the use of virtualization we can use resources to run many programs, i.e. all programs will behave as if they have the whole computer to themselves.

The simple example of virtualization is as follows: we have a file `cpu.c` as well as `common.h` header file.



```
common.h x cpu.c x
1  #ifndef __common_h__
2  #define __common_h__
3
4  #include <sys/time.h>
5  #include <sys/stat.h>
6  #include <assert.h>
7
8  double GetTime() {
9      struct timeval t;
10     int rc = gettimeofday(&t, NULL);
11     assert(rc == 0);
12     return (double) t.tv_sec + (double) t.tv_usec/1e6;
13 }
14 void Spin(int howlong) {
15     double t = GetTime();
16     while ((GetTime() - t) < (double) howlong)
17         ; // do nothing in loop
18 }
19 #endif
```



```
common.h x cpu.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "common.h"
4
5  int main(int argc, char *argv[])
6  {
7      if (argc != 2) {
8          fprintf(stderr, "use: cpu <string>\n");
9          exit(1);
10     }
11     char *str = argv[1];
12
13     while (1) {
14         printf("%s\n", str);
15         Spin(2);
16     }
17     return 0;
18 }
```

In order to run the code, the user should enter the command in the terminal to first compile it. We should as well supply the parameter, in this case, `string`. Otherwise, the output will be “use: cpu <string>”. As there is a `Spin(2)` function, the program will wait for two seconds before executing again. This is an example of running a single program. In order to stop the process we should press `Ctrl+C`.

```

darina@darina-VirtualBox:~/Desktop/OperatingSystemsProject
$ gcc -o cpu cpu.c -Wall
darina@darina-VirtualBox:~/Desktop/OperatingSystemsProject
$ ./cpu
use: cpu <string>
darina@darina-VirtualBox:~/Desktop/OperatingSystemsProject
$ ./cpu ProcessA
ProcessA
ProcessA
ProcessA
ProcessA
ProcessA
ProcessA
ProcessA
ProcessA
^C
darina@darina-VirtualBox:~/Desktop/OperatingSystemsProject
$

```

In order to run the program on background, we add an ampersand &. The output will be printing over and over again every two seconds. The difference is that we can simultaneously do something else in the command line, but the output will keep printing. In order to stop the process we should kill it. We can open another terminal and execute following command.

```

darina@darina-VirtualBox:~$ pidof ./cpu
11681
darina@darina-VirtualBox:~$ kill -9 11681

```

And we can see on the initial terminal that the process was killed.

```

      PID TTY          TIME CMD
    6442 pts/0        00:00:00 bash
    11763 pts/0        00:00:00 ps
[2]+  Killed                  ./cpu ProcessA
darina@darina-VirtualBox:~/Desktop/OperatingSystemsProject
$

```

We can also run two copies of the same program by using the command:

```

darina@darina-VirtualBox:~/Desktop/OperatingSystemsProject
$ ./cpu ProcessA & : ./cpu ProcessB &

```

We can see that the operating system is able to run two programs at the same time. In order to stop the programs, we kill both processes.

- Unix Process API

Creation of a process, single inter process relation.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <string.h>
6
7  int main() {
8      printf("My initial pid: %d \n", (int) getpid());
9      int rc = fork();
10     if(rc < 0){
11         fprintf(stderr, "Cannot create process\n");
12         exit(1);
13     }
14     if(rc == 0){
15         printf("I am child, my pid: %d \n", (int) getpid());
16         char *myargs[3];
17         myargs[0] = strdup("wc");
18         myargs[1] = strdup("process4.c");
19         myargs[2] = NULL;
20
21         execvp(myargs[0], myargs);
22         printf("This will not print");
23     } else {
24         int rc wait = wait(NULL);
25         printf("I am parent, my pid: %d \n", (int) getpid());
26     }
27     return 0;
28 }

```

```

darina@darina-VirtualBox:~/Desktop/OperatingSystemsProject
$ ./process4
My initial pid: 12727
I am child, my pid: 12728
27 82 600 process4.c
I am parent, my pid: 12727

```

getpid() returns the process ID of the calling process. The ID is guaranteed to be unique and is useful for constructing temporary file names. getpid() is included in <unistd.h> library. After every compilation and running, the pid number is different. We create a process within the process by using the function fork(). The new process (child process) is a copy of the calling process (parent process) except that the child process has a unique process ID. wait() - a process calling this starts waiting for some other process to finish. It is included in <sys/wait.h> library. wait(NULL) waits for the child process to finish. We use execvp() function to load and execute some code. The parameters should end in NULL. The line after execvp() will not print, because the child process gives up its own code, loads some of the code from some other program, and executes that code.

Part 2

Scheduling Algorithm

CPU Scheduling is a process of determining which process will own the CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least selects one of the processes available in the ready queue for execution. A Scheduling Algorithm is an algorithm that tells us how much CPU time we can allocate to the processes.

We used 3 types of process scheduling algorithms in our project:

1) First Come First Serve (FCFS)

First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when the CPU becomes free, it should be assigned to the process at the beginning of the queue.

2) Shortest Remaining Time

The full form of SRT is Shortest remaining time. It is also known as SJF preemptive scheduling. In this method, the process will be allocated to the task, which is closest to its completion. This method prevents a newer ready state process from holding the completion of an older process.

3) Round Robin Scheduling

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation-free execution of processes.

Context Switching

Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

Virtual and Physical memory and address translation using a page table

Virtual addresses are used by the program executed by the accessing process, while physical addresses are used by the hardware, or more specifically, by the random-access memory (RAM) subsystem. The page table is a key component of virtual address translation that is necessary to access data in memory. A virtual memory address comes in and needs to be translated to the physical address