ECSE 427 Written Assignment 1

Aleksas Murauskas-260718389

Ege Odaci-260722818

Fouad Bitar-260719196

Question 1: A system call is used by a user program to request services from the kernel of the OS it is executed on. An interrupt is triggered by external components that do not have an operating system. Interrupts are handled by the kernel as well.

Question 2: I/O instructions are privileged because : Users may interfere with each other while accessing an input device if there is more than one user on a computer. If instructions are not interfered with each other, they are stored in the kernel mode. For example, if a user is accessing an I/O device such as printer, the other users are not allowed to access the printer. If I/O instructions were running in the user mode and more than one people try to access the device at the same time with different pages to print, then interference would occur between the two processes.

Question 3:

a) Disable all interrupts

This should only be allowed in kernel mode. Say a malicious application or hacker wants to mess with the execution of other programs running (which may further cause security issues), then disabling interrupts will do so as for example applications will not be able to call on the kernel with interrupts.

b) Read the time-of-day clock

This should be allowed by both the user and kernel, considering it is only a read operation and therefore the user should not interrupt the CPU for such a trivial task.

c) Set the time-of-day clock

This should only be allowed by the kernel since malicious programs may want to abuse this and change it.

d) Change the memory map

This should only be allowed by kernel. The memory map lays out where everything is stored, of course users should not be able to change this as it will allow malicious hackers to understand where things are stored and retrieve information in this way. That is why we use as an extra security measure random memory mapping.

Question 4: Answer: A trap is a software interrupt that synchronously transfers control, created by an error or user program exception like dividing by zero or trying to access an invalid memory address. A trap will often switch to kernel mode, and the operating system performs an action to try to resolve the issue. If a trap occurs during a kernel process it may result in a fatal error.

Question 5: Operating system must provide reasonable response time. It provides that response time with a virtual memory technique. Virtual memory allows the execution of a process which is not completely in the memory. This technique allows users to run programs that are larger than physical memory. On the other hand, the system should ensure execution in right order by providing mechanisms for process synchronization and communication. In this way processes don’t stuck on waiting each other forever.

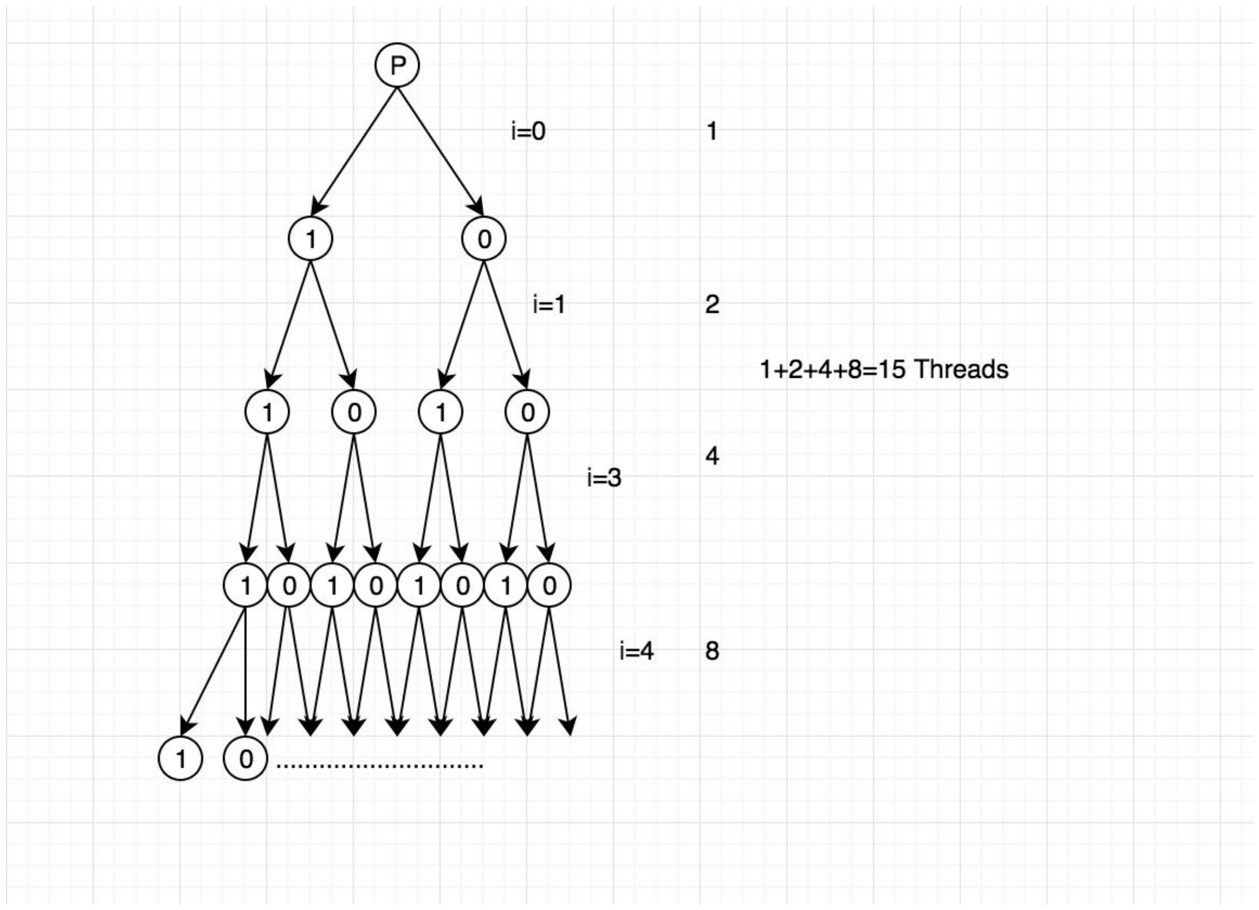
Question 6:

- Since all services are in the same address space in a Monolithic OS, it could be possible for any of the services to access other services and possibly damage the memory of other parts due to bugs in code. Whereas in the microkernel architecture, the services are all separated and each have their own memory space.
- Since the services are separated in the microkernel architecture, if a service crashes then it will not affect the entire system or the microkernel itself, it will crash on its own and be re-started.
- The reason why microkernel architecture can be less reliable is because with the separation of all the services, we need to make more calls to access inter process communication. As a result there is a need for more complex inter process communication mechanisms. At the same time though, the microkernel is simpler than what the privileged part would be in the monolithic, meaning it would have a simpler design less prone to bugs and mistakes.

Question 7: Answer: Input side Processing: 100 ms, disk access 900ms, cache 100ms

All calls use input side processing, and the best case is cache so 200 ms, the worst case is 1000 ms. Best case response rate = 1/.200 = 5 req/s , worst case response rate 1/1 =1 req/s.

Question 8:



Question 9:

- **What will be the contents of the temp2.txt file?**

The output will be both "Message from A" and "Message from B". The order is unknown and could either be one or the other first or second.

- **Modifications to remove "Message from A"**

To remove the print but keep the forked child we would add a close(1) to close the fd pointing to the file temp2.txt so that it cannot write to it.

- **Modifications to ensure it shows up on standard output**

Would have to dupe the standard output before we connect fd 1 to the file temp2.txt

```
> fd = dup(1);
> close(1);
> fd = open(tmp2.txt, …);
> if (fork() == 0) {
>           close(1);
>           dup(fd);
>           printf("Message from A");
```

> }

Question 10: Answer: $s = singular\ time,\ p = parallel\ time$ $\quad$ $the\ speed\ up\ is \frac{800}{290} = 2.76$

$s + p = 800,\ s + \frac{p}{4} = 290, \quad \frac{3p}{4} = 510, \quad p = 680s, \quad s = 120s$

$\frac{680}{800} = 85\%\ parallel$

1 core	2 Core	4 Core	8 Core	16 Core
800 s	460	290 s	205s	162.5s

Question 11: Tee command reads the input and writes it to both and one or many files. Pseudo code for implementing pipe is :

```
tee{
pid = fork();
if(pid ==0) {
close(write);
dup2(read,STDIN_FILENO);
close(read);
}
else{
close(read);
dup2(write,STDOUT_FILENO)
close(write);
}
}
```

Pipe Redirection: Tee command is used like :

```
do_something | tee -a logfile
```

Tee will take one input file and produce output and output file called logfile. For example : ‘echo "Hello" |tee -a logfile.txt’ this will output “Hello” to the screen and also fill the logfile.txt with the word “Hello”.

Question 12:

- Exec

It could fail if permission to the file you are trying to access is denied.
It could fail if the file does not exist.
The new process requires more memory than is allowed by the limit set with setrlimit.

- Fork

The fork system call could fail if there is not sufficient memory to copy the parent's page tables. Another instance of when it might fail would be when the RLIMIT is set restricting the amount of memory allowed for the parent process. Could also have reached the maximum number of user processes allowed. It would also fail if the parent process is consuming too much memory (i.e. more than 50% of system memory.