Assignment 2

Satyaki Ghosh
10077685
T #01

1. DMA stands for Direct Memory Access and it is a used for bulk data movement such as disk I/O it allows processes to access the main memory, independent of the CPU. Multiprogramming is a form of parallel processing in which several programs run at the same time on the same processor. Multiprogramming would not be practical on a computer that does not support DMA because if every process must wait for the CPU to access the main memory then it slows down all the processes and not all the processes are running in parallel or at all anymore.

2. It is not essential that a wrapper of a system call is name the same as the underlying system call, e.g. printf() calls the write() system call. After the read() function in libc is called by the user:
   I. The code for read is put into the register
   II. Trap to the kernel
   III. Dispatch and System call handler is invoked
   IV. Then the process returns to the caller

3. No, it is not possible for a process to go from a blocking state to a running state. This is because a blocking state means that the process is waiting for some event to occur, after the event occurs it must give the ready signal to the CPU to tell it that the process is ready to be executed again. Thus, a process can only go to a ready state from a blocked state before running again.
   No, it is not possible for a process to go from a ready state to a blocking state. This is because if a process is already in a ready state then it is ready to be executed. Thus, a process must always go from a ready state to a running state before it can go to a blocking state.

4. Context switching allows the illusion of sharing a single CPU or limited number of CPUs among many processes. If the number of CPUs is less than the number of process then context switching is needed to implement multitasking. The operating system maintains a context or state for each process, usually in some form of PCB. When OS switches between process A and B, OS saves the A's state in A's PCB and restores B's state from B's PCB.

5. Program in file scan.sh

6. Program in file scan.c

7. Running strace -c and time commands on the bash script gave us these results:

% time	seconds	usecs/call	calls	errors	syscall
68.37	0.005121	1024	5	1	wait4
6.34	0.000475	14	34	13	open
6.13	0.000459	23	20	3	stat
4.55	0.000341	85	4		clone
2.54	0.000190	6	33	6	close
1.94	0.000145	6	23		mmap
1.71	0.000128	6	20		fstat
1.62	0.000121	6	19		rt_sigprocmask
1.58	0.000118	7	16		rt_sigaction
0.69	0.000052	6	9		read
0.56	0.000042	8	5	1	access
0.45	0.000034	11	3		pipe
0.43	0.000032	6	5		getegid
0.36	0.000027	5	5		getuid
0.36	0.000027	5	5		geteuid
0.35	0.000026	5	5		getgid
0.31	0.000023	6	4		lseek
0.29	0.000022	7	3	2	ioctl
0.24	0.000018	5	4		brk
0.23	0.000017	6	3	1	fcntl
0.16	0.000012	6	2		getpid
0.15	0.000011	11	1		getppid
0.15	0.000011	6	2		prlimit64
0.12	0.000009	9	1		uname
0.12	0.000009	9	1		sysinfo
0.11	0.000008	8	1		dup2
0.08	0.000006	6	1		rt_sigreturn
0.08	0.000006	6	1		getpgrp
0.00	0.000000	0	8		mprotect
0.00	0.000000	0	1		munmap
0.00	0.000000	0	1		execve
0.00	0.000000	0	1		arch_prctl
100.00	0.007490		246	27	total

```
real    0m0.009s
user    0m0.003s
sys     0m0.008s
```

Running strace -c and time commands on the C script gave us these results:

% time	seconds	usecs/call	calls	errors	syscall
78.29	0.000238	17	14		stat
10.86	0.000033	6	6		write
10.86	0.000033	33	1		wait4
0.00	0.000000	0	3		read

```
 0.00    0.000000           0         2           open
 0.00    0.000000           0         4           close
 0.00    0.000000           0         4           fstat
 0.00    0.000000           0         6           mmap
 0.00    0.000000           0         4           mprotect
 0.00    0.000000           0         1           munmap
 0.00    0.000000           0         4           brk
 0.00    0.000000           0         1         1 access
 0.00    0.000000           0         1           clone
 0.00    0.000000           0         1           execve
 0.00    0.000000           0         1           fcntl
 0.00    0.000000           0         1           arch_prctl
 0.00    0.000000           0         1           pipe2
------ ----------- ----------- --------- --------- ----------------
100.00    0.000304                    55         1 total

real    0m0.009s
user    0m0.001s
sys     0m0.005s
```

These results are different because a C program makes system calls, while a bash script forks processes to achieve the same goal. System calls are usually more efficient and forking and starting new processes.

8. Program in file sum.cpp