Indian Institute of Information Technology, Kalyani
Mid-Semester Examination 2020

Subject: Operating System *Time: 1 hr 30 min*
Paper Code: CS-401 *Full Marks:* **45**

Instructions : **There are four (4) questions. Answer all of them.**

1. *Give very brief answer to each question:* **[12 × 2]**

   (a) *Why all CPU instructions are not available to the user program?*

   Ans. *CPU instructions such as I/O, memory mapping, interrupt etc. are not available to the user. This is to protect one user from another and also to protect the OS.*

   (b) *How is the address of the first instruction loaded in the program counter (PC) when a computer is switched on?*

   Ans. *The address of the first instruction is pre-determined and is loaded by the power-on hardware.*

   (c) *Name four (4) different regions of the logical address space of a process (Unix/Linux).*

   Ans. *Text or code, data, heap, stack.*

   (d) *Name two different events that may change the state of a process from the running state to the ready state.*

   Ans. *(i) A timer interrupt, (ii) an event that has transferred a higher priority process from the suspended state to ready state.*

   (e) *Where is the program counter (PC) (the address of the next instruction) usually saved during a function call and during a system call?*

   Ans. *In case of a function call it is saved on the user stack (or some register if the function is not calling any other function). In case of a system call it is saved in the kernel stack of the caller process.*

   (f) *Name two events that changes the mode of the CPU from user to kernel (privileged).*

   Ans. *(i) a system call, (ii) a hardware interrupt.*

   (g) *Explain the parameters of* `execve()`.

   ```
   int execve(const char *a, char *const b[], char *const c[]);
   ```

   Ans. `a:` *path of the executable file to be loaded,* `b:` *the command line arguments of* `a`, `c:` *environment of execution.*

   (h) *Why is it necessary to save the content of all CPU registers during a context switching?*

   Ans. *User program has no control over the time of context-switch. OS has no knowledge about the register usage by a user program. So during a context-switching the content of all CPU registers (available to a user process) are saved for the current process and the registers are loaded with values of the scheduled process.*

   (i) *What event initiates a context switching in a time sharing system?*

   Ans. *A timer sends an interrupt to the CPU that transfers control from the user code to the kernel code. The interrupt service routine in tern invokes the scheduler.*

(j) *Will there be a state transition of a running process if it tries to access a valid address not present in the main memory?*

Ans. *The process will go to the suspended state until the block containing the addressed location is loaded in the main memory from disk.*

(k) *Why does the following code in* `main()` *gives a segmentation fault?*

```
int *p = (int *)main;
cin >> *p;
```

Ans. *The region of memory pointed by* `p` *does not have write permission. So it gives memory fault.*

(l) *Why is it necessary to close the write descriptor of a pipe (Linux/Unix) in a process reading from the pipe?*

Ans. *When the writing process of a pipe finishes, it closes the write descriptor. This is suppose to send an EOF signal to the reading process. But that will happen only if all write descriptors of the pipe are closed. The EOF will not be generated if the write descriptor of the reading process remains open.*

2. **[3+4]**

(a) How many child processes are created by the following code for `n = 10`*? Give an expression for the number of child processes in terms of* n*.*

```
void createProc(int n){
     for(int i=1; i<=n; ++i)
        if(!fork()){
          cout << "cpid: " << getpid() << endl;
          exit(0);
        }
     if(n) createProc(n-1);
}
```

Ans. *The function call creates* $1+\cdots+n = n(n+1)/2$ *child processes. So for* $n = 10$*, 55 child processes are created.*

(b) Write a **recursive** *function* `void createNproc(int n)` *that creates* $n(\geq 0)$ *child processes.*

Ans.

```
void createProc(int n){
     if(n==0) return;
     if(!fork()){
        cout << "cpid: " << getpid() << endl;
        exit(0);
     }
     createProc(n-1);
}
```

3. **[3+4]**

(a) Consider the line '`// **`' in the following code.

```
int main(){
    char *str = (char *)"IIIT Kalyani\n";
    int fd;

    write(1, str, strlen(str));
    fd = dup2(1, 3);   // **
    write(3, str, strlen(str));
    cout << "File Desc: " << fd << endl;
    return 0;
}
```

Replace it by x86-64 inline assembly code `__asm__ __volatile__();` *The system call code for* `dup2()` *is 33, to be loaded in* `rax`*. The return value from* `eax (=a)` *should go to* `fd`*. The first and the second parameters are passed through* `rdi` *and* `rsi` *respectively.*

Ans.

```
__asm__ __volatile__(
        "movq $33, %%rax \n\t"
        "movq $1,  %%rdi \n\t"
        "movq $3,  %%rsi \n\t"
        "syscall         \n\t"
        :"=a" (fd)
    );
```

(b) Write the essential part of C++ code (using `dup()`*,* `close` *and* `dup2()`*), that writes in the file* `outData` *using* `cout`*. But once the writing is complete, it should restore* `stdout` *in its proper file descriptor.*

Ans.

```
int main(){
    char *str = (char *)"IIIT Kalyani\n";

    cout << "In stdout: " << str;
    dup(1);
    close(1);
    open("./outData", O_WRONLY | O_CREAT, 0777);
    cout << "In file: " << str;
    dup2(3,1);
    cout << "In stdout again: " << str;

    return 0;
}
```

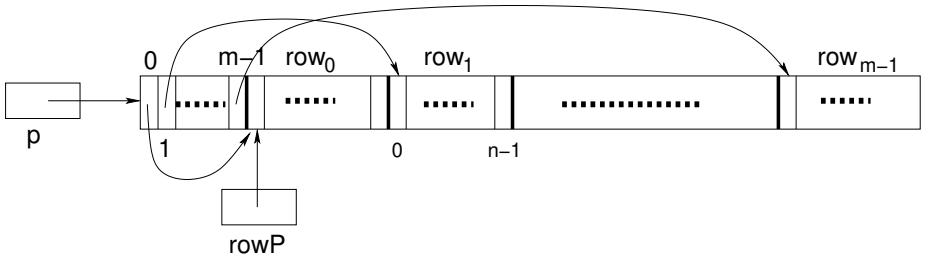
4. **[3+4]**

(a) Draw a diagram and explain the data structure created in the shared memory by the following code.

```
int main(){
    int m, n, size, **p, shmID, *rowP;
    struct shmid_ds buff;

    cin >> m >> n;
    size = m*(sizeof(int *)+n*sizeof(int));
    shmID = shmget(ftok("./", 1234), size, IPC_CREAT | 0777);
    p = (int **) shmat(shmID, 0, 0777);
    rowP = (int *)(p+m);
    for(int i=0; i<m; ++i) {
       p[i] = rowP;
       rowP = rowP+n;
    }
    shmdt(p) ;
    shmctl(shmID, IPC_RMID, &buff);
    return 0;
}
```

Ans.



The program creates a 2-D array like structure in the shared memory by allocating m `int *` *pointers and $m \times n$* `int` *locations. The number of rows are m and the number of columns are n. The array can be accessed as* `p[i][j]` *where $0 \leq i < m$ and $0 \leq j < n$.*

(b) Write C++ code (only the relevant portions) to allocate two 1-D array of type `int` *of sizes* `p` *and* `q` *in a shared memory. Positive integers* `p, q` *are input to the program.*

Ans.

```
int main(){
    int m, n, size, *p, *q, shmID;
    struct shmid_ds buff;

    cin >> m >> n;
    size = (m+n)*sizeof(int);
    shmID = shmget(ftok("./", 1234), size, IPC_CREAT | 0777);
    p = (int *) shmat(shmID, 0, 0777);
    q = p+m;
    cout << hex << p << " " << q << endl;

    shmdt(p) ;
    shmctl(shmID, IPC_RMID, &buff);
    return 0;
}
```