1. (1 pt)
   - Running state – a process is running on CPU.
   - Ready state – a process waiting for CPU (shot term scheduler will assign CPU for it)
   - Blocked state – a process waiting for some I/O (signal, child termination, input from keyboard ..)
   - Transaction 1 – a process is suspended since it need some I/O
   - Transaction 2 – a process used up it's time quantum (time out)
   - Transaction 3 – since CPU become available, CPU scheduler select a process from ready queue and let it use CPU
   - Transaction 4 –some I/O become available; a process ready to be selected by scheduler

2. (3 pt.)

```
/* task2.c: demonstrate waitpid system call */
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
int main()
{
        pid_t pid, pid1, ppid;
        int i, endID, status;

        pid = fork(); /*create the first child */
        if (pid == 0) /* code for first child */
        {
                //create a gramd child by the first child
              pid1 = fork();
                if (pid1 >0)
                {
                        for (i=0; i<100; i++)
                        {
                                printf("I am your child with ID = %d \n", getpid());
                                sleep(1);
                        }
                        _exit(0);
                }
                else
                {
                        ppid = getppid();
                        while (1)
                        {
                                if (getppid()==ppid)
                                {
                                        printf("I am your grandchild \n");
                                        sleep(1);
                                }
                                else
                                        _exit(0);
                        }
                }
        }
        else
        {
                while (1)
                {
                        endID=waitpid(pid, &status, WNOHANG|WUNTRACED);
                        if (endID==0) //child still running
                        {
                                printf("I am your parent with ID= %d\n",getpid());
                                sleep(1);
                        }
                        else
                        {
                                printf("Now my job is over \n");
                                exit(0);
                        }
                }
        }
}
```

3. (2 pt.)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (int argc, char *argv[])
{
      int i,n,m ;
      char * buffer, ch[20];

      i=atoi(argv[1]);

      buffer = (char*) malloc (i+1);
      if (buffer==NULL)
            exit (1);

      for (n=0; n<i; n++)
            buffer[n]=rand()%26+'a';
      buffer[i]='\0';
      printf ("Random string: %s\n",buffer);

      write (1,"integer for extension?", 22);
      read(0, ch, 20);
      m=atoi(ch);
      buffer = (char*) realloc(buffer, (i+m)*sizeof (char));

      for (n = i;  n< i+m; n++)
            buffer[n]=rand()%26+'a';
       buffer[i+m]='\0';
        printf ("Extended Random string: %s\n", buffer);

      free (buffer);

      return 0;
}
```

4.

a. (0.5 pt.) What is Race condition? –A situation where two or more processes are reading or writing some shared data and the final result depends on who runs precisely when.

b. (0.5 pt.) What is mutual exclusion of critical section – only one process can access shared resources at any moment.

c. (0.5 pt.) What is Zombie process – when a child process terminate if parent does not call wait() or waitpid() to get child process's termination status, child will be remain as a zombie.

d. (0.5 pt.) A process can create a child process by using fork() or vfork() system call. Discuss two main differences between two child created by fork() and vfork().

A child with fork(): has it's own address space. Only share text segment with its parent.
A child with vfork(): memory space is shared with its parent. A parent always wait for the child.

5. (2 pt.)

```
#include  <stdio.h>
#include  <stdlib.h>
#include <unistd.h>

int  main(void)
{
     int pid;
     pid=fork();
     if(pid>0)
         while (1);
     else
     {
            pid =fork();
            if (pid >0)
                   exit(0);
            else
                while(1);
     }
}
```