1.

```
#include  <stdio.h>
#include  <stdlib.h>
int  main(void)
{
     int pid;
     pid=fork();
     if(pid>0)
     {
         while (1)
          ;
     }
     else
     {
           exit(0);
     }
}
```

2.

- Running state – a process is using CPU for calculation
- Ready state – a process waiting for CPU
- Blocked state – a process waiting for I/O finish
- Transaction 1 – a process need I/O
- Transaction 2 – a process time out
- Transaction 3 – scheduler select a process to run
- Transaction 4 – I/O become available to run

3.

```
/* task3.c */
#include <stdio.h>
#include <unistd.h>

extern char **environ;

int main(int argc, char *argv[])
{
     char **p = environ;

     while (*p != NULL)
    {
        printf("%s\n", *p);
        *p++;
    }

     return 0;
}
```

4.

a.

A situation where two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called race condition.

b.

Since every process must have a parent, systemd (pid =1) becomes its parent and runs

5.

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void hd1 (int sig)
{
        exit(0);
}
void hd2 (int sig)
{
        kill (getppid(), SIGUSR2);
        _exit(0);
}
int main()
{
        pid_t pid, pid1;
        int i, count;
        if ((pid = fork()) <0)
        {
                perror ("fork error");
                exit (1);
        }
        if (pid == 0)/* child process */
        {
                if ((pid = fork()) <0)
                {
                        perror ("fork error");
                        exit (1);
                 }
                if (pid > 0)
                {
                        signal (SIGUSR1, hd2);
                        while (1)
                        {
                                printf("This is child\n");
                                sleep(1);
                        }
                }
                else // grandchild process
                {
                        count =1;
                        while (1)
                        {
                                if (count == 10)
                                        kill (getppid(), SIGUSR1);
                                if (getppid() == 1)
                                        break;
                                printf("grand child \n");
                                sleep(1);
                                count++;
                        }
                }
        }
        else /* parent process */
        {
                signal (SIGUSR2, hd1);
                while (1)
                {
                        printf("This is parent\n");
                        sleep(1);
                }

        }

 }
```

6.

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

int main()
{
    int pfd[2], pfd1[2];  // pipe file descriptor
    char buf[80];

    pipe(pfd);
    pipe(pfd1);
    if (fork() > 0)
    {
        close(pfd[0]);
         close(pfd1[1]);
        write(pfd[1], "I love you my child!", 20);
        read(pfd1[0], buf, 20);
        printf("My child said %s\n", buf);
        exit(0);
    }
    else
    {
         close(pfd[1]);
         close(pfd1[0]);
        read(pfd[0], buf, 20);
        printf("My mom said %s\n", buf);
       write(pfd1[1], "I love you too", 15);
     }

  return 0;
}
```

7.

- 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.
- Mutual exclusion of critical section – only one process can access shared resources at any moment.
- Zombie process – a child process terminate without calling wait or waitpid() system call by parent process
- What are five components of a C program memory layout?
  Text segment, initialized data segment, uninitialized data segment, stack, queue

8.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void Bye(void *);
void Bye1(void *);
void *Thread1(void *);
void *Thread2(void *);
int main()
{
  pthread_t tid0, tid1, tid2;
   int rc=0;
   void *tmp1, *tmp2;
   tid0=pthread_self();
   rc = pthread_create(&tid1, NULL, Thread1, (void *)tid0);
   rc = pthread_create(&tid2, NULL, Thread2, (void *)tid1);
       pthread_cleanup_push(Bye1, NULL);
    while (1) { pthread_testcancel();
           printf("In the original thread\n");
       sleep(1);
    }
       pthread_cleanup_pop(0);
}
void Bye1(void *arg)
{
  printf("END OF PROGRAM\n");
  exit(0);
}
void Bye(void *arg)
{
  pthread_t tid = (pthread_t)arg;
  int rc = pthread_cancel(tid);
  printf("BYE!\n");
}

void *Thread1(void *parm)
{
  pthread_t tid = (pthread_t)parm;
  pthread_cleanup_push(Bye, (void *)tid);
  while (1) { pthread_testcancel();
        printf("In the first thread\n");
    sleep(1);
  }
  pthread_cleanup_pop(0);
}
void *Thread2(void *parm)
{
  int count, rc;
  count =0;
  pthread_t tid = (pthread_t)parm;

  while (1){count++;
       printf("In the second thread\n");
       sleep(1);
       if (count == 10)
              rc = pthread_cancel(tid);
       }
}
```

9.

- **standardIO.c**

```
//stanardIO.c
#include <stdio.h>

#define MAXLINE 80
int main()
{
    char line[MAXLINE];
    int  size;

   while ( (size = read(0, line, MAXLINE)) > 0)
   {
         write(1, line, size);
   }
   return 0;
}
```

- (10 pt.) **popensum.c** (**DO NOT USE GLOBAL VARIABLE)**

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

#define MAXLINE 80

int main(void)
{
      char  line[MAXLINE];
      int int1, int2, n;
      FILE *fpin; /* use pointer to read from pipe */

      if ((fpin = popen("./standardIO", "r")) == NULL)
            perror("popen error");

      while (fgets(line, MAXLINE, fpin) != NULL)
      {
            /* chose first two string as two integer */
            if (sscanf(line, "%d%d", &int1, &int2) == 2)
            {
                  sprintf(line, "%d\n", int1 + int2);
                  n = strlen(line);
                  if (write(STDOUT_FILENO, line, n) != n)
                        perror("write error");
            }
            else /* if first two string is not integer */
                  printf("invalid input\n");


            }//end of while loop

      if (pclose(fpin) == -1)
      perror("pclose error");
      exit(0);
}
```

10.

```
/* malloc example: string generator*/
#include <stdio.h>
#include <stdlib.h>

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

  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);
  free (buffer);

  return 0;
}
```

11.

```
#include  <stdio.h>
#include  <string.h>
#include <unistd.h>
#include  <sys/types.h>

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