1. (20 pt.)

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

void *thread1(void *);
void *thread2(void *);
void *thread3(void *);
int main (int argc, char *argv[])
{
      int rc;
      pthread_t tid1, tid2, tid3;
      void *tret1, *tret2, *tret3;

      if (argc ==1)
      {
            perror("argument number error");
            exit (1);
      }
      int num = atoi(argv[1]);


      rc=pthread_create(&tid1, NULL, thread1, (void *) num);
      rc =pthread_join(tid1, &tret1);
      rc=pthread_create(&tid2, NULL, thread2, (void *) tret1);
      rc =pthread_join(tid2, &tret2);
      rc=pthread_create(&tid3, NULL, thread3, (void *) tret2);
      rc =pthread_join(tid3, &tret3);
      printf ("the final result of three theads's calcuation is %d\n",
(int) tret3);
      exit (0);
}
void *thread1(void *arg)
{
      int num = (int)arg;
      num = num +10;
    return ((void *) num);
}

void *thread2(void *arg)
{
      int num = (int)arg;
      num = num +20;
      return ((void *) num);
}

void *thread3(void *arg)
{
      int num = (int)arg;
      num = num +30;
      return ((void *) num);
}
```

2. (20 pt.)

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

void terminated (int sig)
{
      if (sig == SIGUSR1)
      {
            printf("Mom said I need study! Bye! I need go! \n");
            _exit(0);// child send SIGCHLD when it is terminated
      }
      else if (sig == SIGCHLD)
      {
            printf("Please don't disturb my child studying! \n");
             _exit(0);
      }
}

int main()
{
   pid_t pid;
   int i, count;

   if ((pid = fork()) <0)
   {
      perror ("fork error");
      exit (1);
   }
   if (pid >0)/* parent process */
   {
      count =0;
      while (1)
      {
              signal (SIGCHLD, terminated);
              printf("Son! It is time to study\n", count+1);
              sleep(1);
              if (count==10)
                     kill(pid, SIGUSR1); //send SIGUSR1 to child
              count++;
      }
     }


   else /* child process */
   {
     while (1)
     {
          printf("Mom! I want play forever!\n");
          signal (SIGUSR1, terminated);
          sleep(1);
     }
    }
  }
```

3. .

a. (10 pt.)

```
//twosum.c
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXLINE 256

int main(void)
{
      char  line[MAXLINE];
      int int1, int2, n, size;

      while ((size = read(0, line, MAXLINE))>0) //read from stdin
      {
            /* 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(1, line, n) != n)
                        perror("write error");
            }
            else /* if first two string is not integer */
                  printf("invalid input\n");

      }//end of while loop
      exit(0);
}
```

b. (20 pt.)

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

#define MAXLINE 256

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

      if ((fpin = popen("./twosum", "w")) == NULL)
            perror("popen error");
      int fd = fileno(fpin); // Get the file descriptor for `popen`
      printf("Any input for a child? \n");

      while ((size = read(0, line, MAXLINE))>0)
      {
            n = strlen(line);
            if (write(fd, line, n) != n)
                  perror("write error");
            printf("Any input for a child? \n");
      }//end of while loop

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

4. (25 pt.)

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

int main()
{
    int p1[2];  // Pipe from parent to first child
    int p2[2];  // Pipe from first child to second child
    int p3[2];  // Pipe from second child to parent
    char buff[80], buff1[80];
    int n;
    pid_t pid1, pid2;
    int input_num, result1, result2;

    pipe(p1); pipe(p2); pipe(p3); // create three pipes

    pid1 = fork();  // Create the first child

    if (pid1 == 0) // Child process 1
    {
         char cbuff[80]; int cn;
         close(p1[1]);  // Close write end of parent-to-child1 pipe
         close(p2[0]);  // Close read end of child1-to-child2 pipe
         cn = read(p1[0], cbuff, 80); // Read input from parent
         printf("Child1 received input from parent: %s\n", cbuff);

         input_num = atoi(cbuff);
         result1 = input_num * 10;// Perform multiplication
         char *tbuff = itoa(result1);
         write(p2[1], tbuff, sizeof(tbuff));// Send result to sibling
         printf("Child 1 sent result to sibling: %d\n", result1);
         exit(0);
    }
    else // parent part
    {
        pid2 = fork();  // Create the second child
        if (pid2 == 0) // Child process 2
        {
               close(p2[1]);  // Close write end of child1-to-child2 pipe
               close(p3[0]);  // Close read end of child2-to-parent pipe
               char cbuff[80];
               read(p2[0], cbuff, 80);// Read result from sibling
               printf("Child 2 received result from sibling:  %s\n", cbuff);

               input_num = atoi(cbuff); // change c-string to integer
               result1 = input_num * 100;// Perform multiplication
               char *kbuff = itoa(result1); //change integer to c-string
               write(p3[1], kbuff, sizeof(kbuff));// Send result to the parent
               printf("Child2 sent result to parent: %d\n", result1);
               exit(0);
         }
         else // parent
         {
              close(p1[0]);  // Close read end of parent-to-child1 pipe
              close(p3[1]);  // Close write end of child2-to-parent pipe
              write(1, "Enter an integer: ", 18);
              n= read (0, buff, sizeof(buff)); //read from stdin
              write(p1[1], buff, n);// Send input to first child
              printf("Parent sent data to a child: %s\n", buff);
              read(p3[0], buff1, 80);// Read result from second child
              printf("Parent received final result: %s\n", buff1);
              exit (0);
         }
    }
       exit (0);
}
```