1.

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

void terminated (int sig)
{
   printf("Mom said \" go away\" \n");
   _exit(0);
}
int main()
{
   pid_t pid;
   int i;
   printf("alarm application start \n");
   if ((pid = fork()) <0)
   {
      perror ("fork error");
      exit (1);
   }
   if (pid >0)/* parent process */
   {
      for (i =0; i<=10; i++)
      {
           printf("This is your Mom \n");
           sleep(1);
      }
      kill (pid, SIGUSR1);
      sleep(1);
      for (i =0; i<=2; i++)
      {
           printf("Sorry my son \n");
           sleep(1);
      }

   }
   else /* child process */
   {
      signal (SIGUSR1, terminated);

     while (1)
     {
          printf("I am Immortal\n");
          sleep(1);
     }

   }

 }
```

2.

```
#include <pthread.h>
#include <stdio.h>
void Bye1(void *);
void Bye(void *);
void *Thread1(void *);
void *Thread2(void *);
int main()
{
        pthread_t tid, tid1, tid2;
        int rc=0;
        void *tmp1, *tmp2;
        tid = pthread_self();

   pthread_cleanup_push(Bye1, NULL);

   rc = pthread_create(&tid2, NULL, Thread1, (void *)tid);
   rc = pthread_create(&tid1, NULL, Thread2, (void *)tid2);
   while (1)
        {
               printf("This is original thread!\n");
               pthread_testcancel();
               sleep(1);
        }
        pthread_cleanup_pop(0);
   return 0;
}

void Bye1(void *arg)
{
        printf("Original thread job is done! \n");
}

void Bye(void *arg)
{
        pthread_t tid = (pthread_t)arg;
        int rc;
        printf("The first thread job is done! \n");
        rc = pthread_cancel(tid);

}

void *Thread1(void *arg)
{
        pthread_t tid = (pthread_t)arg;
        printf("Entered the first thread\n");
        pthread_cleanup_push(Bye, (void *)tid);
        while (1) {
               printf("This is the first thread!\n");
               pthread_testcancel();
               sleep(1);
        }
        pthread_cleanup_pop(0);
}

void *Thread2(void *arg)
{
  int count, rc;
  pthread_t tid = (pthread_t)arg;
  printf("Entered the second thread\n");
  for (count=1; count <=10; count++)
  {
        printf("This is the second thread running %d times\n", count);
        sleep(1);
   }
   rc = pthread_cancel(tid);
   printf("The second thread job is done!\n");

}
```

3.

Child Process:
  Glocal variable: 11
Stack variable: 21

Parent Process:
  Global variable: 15
  Stack variable 25