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

int pid1, pid2;

int main()
{
int fd[2];
char OutPipe[100], InPipe[100];
pipe(fd);
while((pid1=fork())==-1);
if(pid1==0)
{
lockf(fd[1], 1, 0);
sprintf(OutPipe, "Child 1 is sending message!");
write(fd[1], OutPipe, 50);
lockf(fd[1], 0, 0);
sleep(1);
exit(0);
}
else
{
while((pid2=fork())==-1);
if(pid2==0)
{
lockf(fd[1], 1, 0);
sprintf(OutPipe, "Child 2 is sending message!");
write(fd[1], OutPipe, 50);
lockf(fd[1], 0, 0);
sleep(1);
exit(0);
}
else
{
wait(0);
wait(0);
read(fd[0], InPipe, 50);
printf("%s\n",InPipe);
read(fd[0], InPipe, 50);
printf("%s\n", InPipe);
```

```
exit(0);
}
}
return 0;
}
```

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

void* print_a(void*);
void* print_b(void*);

int main()
{
pthread_t t0;
pthread_t t1;

if (pthread_create(&t0, NULL, print_a, NULL)==-1)
{
puts("fail to create pthread t0");
exit(1);
}
if (pthread_create(&t1, NULL, print_b, NULL)==-1)
{
puts("fail to create pthread t1");
exit(1);
}
void * result;
if (pthread_join(t0, &result)==-1)
{
puts("fail to recollect t0");
exit(1);
}
if (pthread_join(t1, &result)==-1)
{
puts("fail to recollect t1");
exit(1);
}
return 0;
}

void* print_a(void* a)
{
for (int i = 0; i < 3; i++)
{
sleep(1);
printf("aa\n");
```

```
}
return NULL;
}

void* print_b(void* b)
{
for (int i = 0; i < 6; i++)
{
sleep(1);
printf("bb\n");
}
return NULL;
}
```

大小写转换：

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <ctype.h>
#define BUFFER_SIZE 25
#define READ_END 0
#define WRITE_END 1
int main(void)
{
  char write_msg[BUFFER_SIZE] = "Greetings";
  char read_msg[BUFFER_SIZE];
  pid_t pid;
  int first_pipe[2];
  int second_pipe[2];
  int i;
  if (pipe(first_pipe)==-1) {
    fprintf(stderr,"First Pipe failed");
    return 1;
  }
    if (pipe(second_pipe)==-1) {
                        fprintf(stderr,"Second Pipe failed");
                        return 1;
              }
  pid = fork();
  if (pid<0) {
    fprintf(stderr, "Fork failed");
    return 1;
  }
  if (pid>0) {    /* parent process */
    /* close the unused ends of each pipe */
    close(first_pipe[READ_END]);
    close(second_pipe[WRITE_END]);
    /* write to the pipe */
    write(first_pipe[WRITE_END],write_msg,25);
    /* now close the write end of the pipe */
    close(first_pipe[WRITE_END]);
    /* read the result from the second pipe */
    read(second_pipe[READ_END],read_msg,25);
    printf("parent read >%s<\n",read_msg);
/* close the read end of the pipe */
    close(second_pipe[READ_END]);
```

```
  }
  else { /* child process */
    /* close the unused ends of the pipes */
    close(first_pipe[WRITE_END]);
    close(second_pipe[READ_END]);
    /* read from the pipe */
    read(first_pipe[READ_END],read_msg,25);
    printf("child read >%s<\n",read_msg);
    /* reverse the string */
    for (i = 0; i < strlen(read_msg); i++) {
      if (isupper(read_msg[i]))
        write_msg[i] = tolower(read_msg[i]);
      else if (islower(read_msg[i]))
        write_msg[i] = toupper(read_msg[i]);
      else
        write_msg[i] = read_msg[i];
    }
    /* write to the pipe */
    write(second_pipe[WRITE_END],write_msg,25);
    /* close the write end of the pipe */
    close(first_pipe[READ_END]);
    close(second_pipe[WRITE_END]);
  }
  return 0;
}
```

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
/* the list of integers */
int *list;
/* the threads will set these values */
double average;
int maximum;
int minimum;
void *calculate_average(void *param);
void *calculate_maximum(void *param);
void *calculate_minimum(void *param);
int main(int argc,char *argv[])
{
  int i;
  int num_of_args = argc-1;
  pthread_t tid_1;
  pthread_t tid_2;
  pthread_t tid_3;
  /* allocate memory to hold array of integers */
  list = malloc(sizeof(int)*num_of_args);
  for (i = 0; i < num_of_args; i++)
    list[i] = atoi(argv[i+1]);
  /* create the threads */
  pthread_create(&tid_1,NULL, calculate_average, &num_of_args);
  pthread_create(&tid_2,NULL, calculate_maximum, &num_of_args);
  pthread_create(&tid_3,NULL, calculate_minimum, &num_of_args);
  /* wait for the threads to exit */
  pthread_join(tid_1, NULL);
  pthread_join(tid_2, NULL);
  pthread_join(tid_3, NULL);
  printf("The average is %f\n", average);
  printf("The maximum is %d\n", maximum);
  printf("The minimum is %d\n", minimum);
  return 0;

}
```

```
void *calculate_average(void *param)

{
  int count = *(int *)param;
  int i, total = 0;
  printf("count = %d\n",count);
  for (i = 0; i < count; i++)
    printf("%d\n",list[i]);
  for (i = 0; i < count; i++)
    total += list[i];
  average = total / count;
  pthread_exit(0);
}
void *calculate_maximum(void *param)
{
  int count = *(int *)param;
  int i;
  maximum = list[0];
  for (i = 1; i < count; i++)
    if (list[i] > maximum)
      maximum = list[i];
  pthread_exit(0);
}

void *calculate_minimum(void *param)
{
  int count = *(int *)param;
  int i;
  minimum = list[0];
  for (i = 1; i < count; i++)
    if (list[i] < minimum)
      minimum = list[i];
  pthread_exit(0);
}
```