#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) //子1进程

{

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) //子2进程

{

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

{ //父进程

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;

}

#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);

}