## Mycat.c

#include<stdio.h>

#include<fcntl.h>

int copy\_file(FILE \*src, FILE \*dest)

{

int c;

while((c = getc(src)) != EOF)

putc(c, dest);

return c;

}

int main(int argc, char\*\* argv)

{

if(argc == 1)

{

copy\_file(stdin, stdout);

}

else

{

char \*filename;

FILE \*infile;

for(int i = 1; i < argc; i++)

{

filename = argv[i];

if((infile = fopen(argv[1], "r")) == NULL)

{

printf("mycat: %s: No such file or directory\n", filename);

continue;

}

copy\_file(infile, stdout);

fclose(infile);

}

}

return 0;

}

## Mycat2.c

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<fcntl.h>

#include<unistd.h>

int copy\_file(int infile, int outfile)

{

int num;

char buf[1];

do{

num = read(infile, buf, 1);

write(outfile, buf, num);

}while(num == 1);

return num;

}

int main(int argc, char\*\* argv)

{

if(argc == 1)

{

copy\_file(0, 1);

}

else

{

char \*filename;

int infile;

for(int i = 1; i < argc; i++)

{

filename = argv[i];

if((infile = open(argv[i], O\_RDONLY)) == -1)

{

printf("mycat: %s: No such file or directory\n", filename);

continue;

}

copy\_file(infile, 1);

close(infile);

}

}

return 0;

}

## Mycp.c

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<fcntl.h>

#include<unistd.h>

int main(int argc, char\*\* argv)

{

int infile, outfile;

char buf[1024];

int num;

if(argc != 3)

{

printf("The format must be:cp file\_src file\_des");

exit(0);

}

if((infile = open(argv[1], O\_RDONLY)) == -1)

{

perror("open1");

exit(0);

}

if((outfile = open(argv[2], O\_CREAT | O\_EXCL | O\_WRONLY, 0644)) == -1)

{

perror("open2");

exit(0);

}

do

{

num = read(infile, buf, 1024);

write(outfile, buf, num);

}while(num == 1024);

close(infile);

close(outfile);

return 0;

}

## Mycp2.c

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<fcntl.h>

#include<unistd.h>

int copy\_file(int infile, int outfile)

{

int num;

char buf[1];

do{

num = read(infile, buf, 1);

write(outfile, buf, num);

}while(num == 1);

}

int main(int argc, char\*\* argv)

{

int infile, outfile;

if(argc != 3)

{

printf("The format must be:cp file\_src file\_des\n");

exit(0);

}

if((infile = open(argv[1], O\_RDONLY)) == -1)

{

printf("mycp: %s: No such file or directory\n", argv[1]);

exit(0);

}

if((outfile = open(argv[2], O\_CREAT | O\_EXCL | O\_WRONLY, 0644)) == -1)

{

printf("mycp: %s: Can't create such file\n", argv[2]);

exit(0);

}

copy\_file(infile, outfile);

close(infile);

close(outfile);

return 0;

}

## Mysys.c

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <string.h>

#define MAX\_BUFFLEN 1024

#define MAX\_NUM 100

int mysys(char \*arg)

{

if(arg[0] == '\0')

return 127;

char code[MAX\_BUFFLEN];

char \*argv[MAX\_NUM]; // no more than 100 arguments

int count = 0; // N.O. of arguments

char \*next = NULL;

char \*rest = code;

strcpy(code, arg);

argv[count++] = code;

while(next = strchr(rest, ' '))

{

next[0] = '\0';

rest = next + 1;

// printf("rest = \"%s\"\n", rest);

if(rest[0] != '\0' && rest[0] != ' ')

argv[count++] = rest;

if(count + 2 > MAX\_NUM)

return 127;

}

argv[count++] = NULL;

// printf("[argv]\n");

// for(size\_t i = 0; i < count; i++)

// printf("\t[%d] %s\n", i, argv[i]);

int pid;

pid = fork();

if(pid == 0)

{

int error = execvp(code, argv);

if(error < 0)

{

perror("execvp");

return 127;

}

else

return 0;

}

int status;

wait(&status);

return status;

}

int main()

{

//char \*argv[] = {"ls", "/", NULL};

//execvp("ls", argv);

mysys("pwd");

mysys("echo ,HELLO WORLD , sdfa sdfadf ss ");

mysys("echo /G");

mysys("echo ,,");

mysys("echo");

mysys("asdfasdf");

printf("----------------------------------------------------------\n");

mysys("echo HELLO WORLD");

printf("----------------------------------------------------------\n");

mysys("ls /");

printf("----------------------------------------------------------\n");

return 0

}

## Mysys2.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <errno.h>

#include <sys/types.h>

#define MAX\_BUFFLEN 1024

#define MAX\_NUM 100

int mysys(const char \*cmdstring)

{

pid\_t pid;

int status = -1;

if (cmdstring == NULL)

return 1;

if ((pid = fork()) < 0)

status = -1;

else if (pid == 0)

{

execl("/bin/sh", "sh", "-c", cmdstring, (char \*)0);

exit(127);

}

else

{

while (waitpid(pid, &status, 0) < 0)

{

if (errno != EINTR)

{

status = -1;

break;

}

}

}

return status;

}

int main()

{

//char \*argv[] = {"ls", "/", NULL};

//execvp("ls", argv);

int res;

res = mysys("");

printf("[Status] %d\n", res);

res = mysys("pwd");

printf("[Status] %d\n", res);

res = mysys("echo ,HELLO WORLD , sdfa sdfadf ss ");

printf("[Status] %d\n", res);

res = mysys("echo /G");

printf("[Status] %d\n", res);

res = mysys("echo ,,");

printf("[Status] %d\n", res);

res = mysys("echo");

printf("[Status] %d\n", res);

res = mysys("asdfasdf");

printf("[Status] %d\n", res);

printf("----------------------------------------------------------\n");

res = mysys("echo HELLO WORLD");

printf("[Status] %d\n", res);

printf("----------------------------------------------------------\n");

res = mysys("ls /");

printf("[Status] %d\n", res);

printf("----------------------------------------------------------\n");

return 0;

}

## Sh3.c

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <sys/stat.h>

#include <fcntl.h>

#define MAX\_BUFFLEN 1024

#define MAX\_NUM 100

char \*home;

char \*dir;

int recover\_in;

int recover\_out;

int fdin, fdout;

int fd[2], fd\_tmp[2];

int flag = -1;

void split(char \*src, int \*argc, char \*\*argv)

{

char code[MAX\_BUFFLEN];

int count = 0; // N.O. of arguments

char \*next = NULL;

char \*rest = code;

strcpy(code, src);

argv[count++] = code;

while(next = strchr(rest, ' '))

{

next[0] = '\0';

rest = next + 1;

// printf("rest = \"%s\"\n", rest);

if(rest[0] != '\0' && rest[0] != ' ')

argv[count++] = rest;

if(count + 2 > MAX\_NUM)

return ;

}

argv[count++] = NULL;

\*argc = count - 1;

}

int mysys(const char \*cmdstring)

{

pid\_t pid;

int status = -1;

if (cmdstring == NULL)

return 1;

if ((pid = fork()) < 0)

status = -1;

else if (pid == 0)

{

dup2(fdin, 0);

dup2(fdout, 1);

close(fdin);

close(fdout);

execl("/bin/sh", "sh", "-c", cmdstring, (char \*)0);

exit(127);

}

else

{

while (waitpid(pid, &status, 0) < 0)

{

if (errno != EINTR)

{

status = -1;

break;

}

}

}

return status;

}

int judge\_buff(char \*buff)

{

//printf("In judge: [%s]\n", buff);

if(buff[0] == '\0')

return 0;

char code[MAX\_BUFFLEN];

strcpy(code, buff);

char \*next = strchr(code, ' ');

if(next != NULL)

next[0] = '\0';

//printf("[code] %s", code);

if(strcmp(code, "cd") == 0)

return 1;

else if(strcmp(code, "exit") == 0)

{

//printf("In judge: [%s]\n", buff);

exit(-1);

}

else

return 0;

}

int cd(char \*buff)

{

int argc = 0;

char \*argv[MAX\_NUM]; // no more than 100 arguments

int count = 0; // N.O. of arguments

split(buff, &argc, argv);

count = argc;

if(count == 1)

{

chdir(home);

dir = getcwd(NULL, 0);

}

else

{

int res = chdir(argv[count - 1]);

dir = getcwd(NULL, 0);

if(res == -1)

{

printf("cd: No such path %s\n", argv[count - 1]);

return -1;

}

}

return 0;

}

int go(char \*buff)

{

int res = judge\_buff(buff);

if(res == 0)

mysys(buff);

else if(res == 1)

cd(buff);

else if(res == -1)

return -1;

return 1;

}

void strip(char \*s)

{

size\_t i;

size\_t len = strlen(s);

size\_t offset = 0;

for(i = 0; i < len; ++i){

char c = s[i];

if(c==0x0d||c==0x0a) ++offset;

else s[i-offset] = c;

}

s[len-offset] = '\0';

}

void strip\_char(char \*s, char bad)

{

size\_t i;

size\_t len = strlen(s);

size\_t offset = 0;

for(i = 0; i < len; ++i){

char c = s[i];

if(c==bad) ++offset;

else s[i-offset] = c;

}

s[len-offset] = '\0';

}

void strip\_dup(char \*s)

{

size\_t i;

size\_t len = strlen(s);

for(i = 0; i < len; ++i)

{

char c = s[i];

if(c == '<' || c == '>')

s[i] = '\0';

}

}

void strip\_pipe(char \*s)

{

size\_t i;

size\_t len = strlen(s);

for(i = 0; i < len; ++i)

{

char c = s[i];

if(c == '|')

s[i] = '\0';

}

}

int go\_dup(char \*buff)

{

char code[MAX\_BUFFLEN];

strcpy(code, buff);

char \*a = NULL;

char \*b = NULL;

a = strchr(buff, '<');

b = strchr(buff, '>');

strip\_dup(code);

if(a != NULL && b != NULL)

{

char \*in = a + 1 - buff + code;

char \*out = b + 1 - buff + code;

strip\_char(in, ' ');

strip\_char(out, ' ');

// printf("[in] %s\n", in);

// printf("[out]%s\n", out);

// printf("[code]%s\n", code);

fdin = open(in, O\_RDWR, 0666);

fdout = open(out, O\_CREAT|O\_RDWR, 0666);

if(fdin == -1)

{

printf("File %s open faild\n", in);

return -1;

}

if(fdout == -1)

{

printf("File %s open faild\n", out);

return -1;

}

return mysys(code);

}

else if(a != NULL)

{

char \*in = a + 1 - buff + code;

strip\_char(in, ' ');

fdin = open(in, O\_RDWR, 0666);

fdout = recover\_out;

if(fdin == -1)

{

printf("File %s open faild\n", in);

return -1;

}

return mysys(code);

}

else if(b != NULL)

{

char \*out = b + 1 - buff + code;

strip\_char(out, ' ');

fdin = recover\_in;

fdout = open(out, O\_CREAT|O\_RDWR, 0666);

if(fdout == -1)

{

printf("File %s open faild\n", out);

return -1;

}

return mysys(code);

}

else

{

fdin = recover\_in;

fdout = recover\_out;

return go(buff);

}

}

int count\_pipe(char \*buff, int loc[])

{

char \*next = buff;

int count = 0;

loc[count++] = 0;

while(next = strchr(next, '|'))

{

//printf("[next] %s\n", next);

next = next + 1;

loc[count++] = next - buff;

}

return count;

}

int pipe\_sys(const char \*cmdstring)

{

pid\_t pid;

pid = fork();

if (pid == 0)

{

if(flag == 0)

{

//printf("[flag] %d\t[code] %s\n", flag, cmdstring);

dup2(fd[1], 1);

close(fd[0]);

close(fd[1]);

execl("/bin/sh", "sh", "-c", cmdstring, (char \*)0);

exit(127);

}

else if(flag == 1)

{

//printf("[flag] %d\t[code] %s\n", flag, cmdstring);

dup2(fd[0], 0);

close(fd[0]);

close(fd[1]);

execl("/bin/sh", "sh", "-c", cmdstring, (char \*)0);

exit(127);

}

else if(flag == 2)

{

//printf("[flag] %d\t[code] %s\n", flag, cmdstring);

dup2(fd[0], 0);

close(fd[0]);

close(fd[1]);

// 输出进入临时管道

dup2(fd\_tmp[1], 1);

close(fd\_tmp[0]);

close(fd\_tmp[1]);

execl("/bin/sh", "sh", "-c", cmdstring, (char \*)0);

exit(127);

}

}

wait(NULL);

//printf("wait once\n");

return 0;

}

int go\_pipe(char \*buff)

{

int res;

char code[MAX\_BUFFLEN];

strcpy(code, buff);

strip\_pipe(code);

int loc[MAX\_NUM];

int count = count\_pipe(buff, loc);

//printf("[debug] count: %d\n", count);

if(count == 1)

{

fdin = recover\_in;

fdout = recover\_out;

return go\_dup(buff);

}

for(int i = 0; i < count; i++)

{

//printf("[debug] %d pipe: %s\n", i, code+loc[i]);

if(flag == 2)

{

dup2(fd\_tmp[0], fd[0]);

dup2(fd\_tmp[1], fd[1]);

close(fd\_tmp[0]);

close(fd\_tmp[1]);

pipe(fd\_tmp);

close(fd[1]);

}

if(flag == 0)

{

close(fd[1]);

}

if(i == 0)

{

flag = 0;

}

else if(i == count - 1)

{

flag = 1;

}

else

{

flag = 2;

}

res = pipe\_sys(code + loc[i]);

}

return res;

}

void find\_last\_dir(char \*\*now)

{

char \*next = NULL;

char \*rest = dir;

//printf("[dir] %s\n", dir);

while(next = strchr(rest, '/'))

rest = next + 1;

if(rest == '\0')

\*now = dir;

else

\*now = rest;

}

void print\_prefix()

{

if(strcmp(home, dir) == 0)

printf("\033[33m%c \033[34;1m~ \033[0m", '>');

//printf("[~]$ ");

else

{

char \*now = NULL;

find\_last\_dir(&now);

printf("\033[33m%c \033[34;1m%s \033[0m", '>', now);

//printf("[!]$ ");

}

}

int main()

{

pipe(fd);

pipe(fd\_tmp);

recover\_in = dup(0);

recover\_out = dup(1);

home = getenv("HOME");

dir = getcwd(NULL, 0);

char buff[MAX\_BUFFLEN];

print\_prefix();

while(fgets(buff, sizeof(buff), stdin))

{

strip(buff);

go\_pipe(buff);

pipe(fd);

pipe(fd\_tmp);

print\_prefix();

}

return 0;

}

## P1.c: 使用2个线程计算PI

#include <stdio.h>

#include <pthread.h>

int N = 1000000;

float worker\_output;

float master\_output;

int sign(int n)

{

if(n % 2 == 0)

return 1;

else

return -1;

}

void \*worker(void \*arg)

{

int i;

for(i = N / 2; i < N; i++)

worker\_output += (float)sign(i) / (2\*i + 1);

printf("worker\_output = %.10f\n", worker\_output);

return NULL;

}

void master()

{

for(int i = 0; i < N / 2; i++)

master\_output += (float)sign(i) / (2\*i + 1);

printf("master\_output = %.10f\n", master\_output);

return;

}

int main()

{

pthread\_t worker\_tid;

float total;

pthread\_create(&worker\_tid, NULL, worker, NULL);

master();

pthread\_join(worker\_tid, NULL);

total = worker\_output + master\_output;

printf("PI = %.10f\n", total \* 4);

return 0;

}

## Pi2c: 使用N线程计算PI

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define N 8

#define NR\_TOTAL 1000000

#define NR\_CPU N

#define NR\_CHILD (NR\_TOTAL/NR\_CPU)

typedef struct param {

int start;

int end;

}Param;

typedef struct result {

float sum;

}Result;

int sign(int n)

{

if(n % 2 == 0)

return 1;

else

return -1;

}

void \*compute(void \*arg)

{

Param \*param = (Param \*)arg;

Result \*result;

float sum = 0;

for(int i = param->start; i < param->end; i++)

sum += (float)sign(i) / (2 \* i + 1);

printf("worker %d = %.10f\n", param->start / NR\_CHILD, sum);

result = malloc(sizeof(Result));

result->sum = sum;

return result;

}

int main()

{

pthread\_t workers[NR\_CPU];

Param params[NR\_CPU];

float total = 0;

for(int i = 0; i < NR\_CPU; i++)

{

Param \*param;

param = &params[i];

param->start = i \* NR\_CHILD;

param->end = (i + 1) \* NR\_CHILD;

pthread\_create(&workers[i], NULL, compute, param);

}

for(int i = 0; i < NR\_CPU; i++)

{

Result \*result;

pthread\_join(workers[i], (void \*\*)&result);

total += result->sum;

free(result);

}

printf("PI = %.10f\n", total \* 4);

return 0;

}

## sort.c: 多线程排序

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <string.h>

#define NUMMAX 100

int nums[NUMMAX];

typedef struct param {

int start;

int end;

}Param;

void show\_nums(int \*arr, const char \*str)

{

printf("[%s]\n", str);

for(int i = 0; i < NUMMAX; i++)

{

if(i == 0)

printf("\t");

else if(i % 10 == 0)

printf("\n\t");

printf("%6d", arr[i]);

}

printf("\n");

}

void generate\_nums()

{

srandom(time(NULL));

for(int i = 0; i < NUMMAX; i++)

nums[i] = rand() % 10000;

}

//快速排序

int findPos(int data[], int low, int high) {

//将大于t的元素赶到t的左边，大于t的元素赶到t的右边

int t = data[low];

while(low < high) {

while(low < high && data[high] >= t) {

high--;

}

data[low] = data[high];

while(low < high && data[low] <=t) {

low++;

}

data[high] = data[low];

}

data[low] = t;

//返回此时t在数组中的位置

return low;

}

void quickSort(int data[], int low, int high) {

if(low > high) {

return;

}

int pos = findPos(data, low, high);

quickSort(data, low, pos-1);

quickSort(data, pos+1, high);

}

//冒泡排序

void bubleSort(int data[], int n) {

int i,j,temp;

for(j=0;j<n-1;j++) {

for(i=0;i<n-j-1;i++) {

if(data[i]>data[i+1]) {

temp = data[i];

data[i] = data[i+1];

data[i+1] = temp;

}

}

}

}

int compare(const void \*a, const void \*b)

{

return (\*(int\*)a - \*(int\*)b);

}

//使用stdlib.h里面的qsort()

void \*sort(void \*arg)

{

Param \*param = (Param \*)arg;

int left = param->start;

int right = param->end;

if(left >= right)

return NULL;

qsort(nums + left, right - left, sizeof(int), compare);

return NULL;

}

void merge(const int left, const int mid, const int right)

{

int temp[NUMMAX];

memcpy(temp, nums, NUMMAX \* sizeof(int));

//show\_nums(temp, "temp");

int s1 = left;

int s2 = mid + 1;

int t = left;

while(s1 <= mid && s2 <= right)

{

if(temp[s1] < temp[s2])

nums[t++] = temp[s1++];

else

nums[t++] = temp[s2++];

}

while(s1 <= mid)

nums[t++] = temp[s1++];

while(s2 <= right)

nums[t++] = temp[s2++];

}

int main()

{

generate\_nums();

show\_nums(nums, "unsort");

pthread\_t worker\_tid;

Param params[2];

params[0].start = 0;

params[0].end = NUMMAX / 2;

params[1].start = NUMMAX / 2;

params[1].end = NUMMAX;

pthread\_create(&worker\_tid, NULL, sort, &params[1]);

//sort(&params[1]);

sort(&params[0]);

pthread\_join(worker\_tid, NULL);

merge(0, NUMMAX / 2 - 1, NUMMAX - 1);

show\_nums(nums, "sorted");

return 0;

}

## pc1.c: 使用条件变量解决生产者、计算者、消费者问题

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#define CAPACITY 4

char buffer1[CAPACITY];

char buffer2[CAPACITY];

int in1, in2;

int out1, out2;

int buffer\_is\_empty(int n)

{

if(n == 1)

return in1 == out1;

else if(n == 2)

return in2 == out2;

else

exit(-1);

}

int buffer\_is\_full(int n)

{

if( n == 1)

return (in1 + 1) % CAPACITY == out1;

else if(n == 2)

return (in2 + 1) % CAPACITY == out2;

}

char get\_item(int n)

{

char item;

if(n == 1)

{

item = buffer1[out1];

out1 = (out1 + 1) % CAPACITY;

}

else if(n == 2)

{

item = buffer2[out2];

out2 = (out2 + 1) % CAPACITY;

}

else

exit(-1);

return item;

}

void put\_item(char item, int n)

{

if(n == 1)

{

buffer1[in1] = item;

in1 = (in1 + 1) % CAPACITY;

}

else if(n == 2)

{

buffer2[in2] = item;

in2 = (in2 + 1) % CAPACITY;

}

else

exit(-1);

}

#define ITEM\_COUNT (CAPACITY \* 2)

pthread\_mutex\_t mutex1, mutex2;

pthread\_cond\_t wait\_empty\_buffer1, wait\_empty\_buffer2;

pthread\_cond\_t wait\_full\_buffer1, wait\_full\_buffer2;

void \*consume(void \*arg)

{

int item;

for(int i = 0; i < ITEM\_COUNT; i++)

{

pthread\_mutex\_lock(&mutex2);

while(buffer\_is\_empty(2))

pthread\_cond\_wait(&wait\_full\_buffer2, &mutex2);

item = get\_item(2);

printf("\033[34m consume item: %c\n\033[0m", item); //蓝色为消费者

pthread\_cond\_signal(&wait\_empty\_buffer2);

pthread\_mutex\_unlock(&mutex2);

}

return NULL;

}

void \*compute(void \*arg)

{

char item;

for(int i = 0; i < ITEM\_COUNT; i++)

{

pthread\_mutex\_lock(&mutex1);

while(buffer\_is\_empty(1))

pthread\_cond\_wait(&wait\_full\_buffer1, &mutex1);

item = get\_item(1);

pthread\_cond\_signal(&wait\_empty\_buffer1);

pthread\_mutex\_unlock(&mutex1);

item += 'A' - 'a';

pthread\_mutex\_lock(&mutex2);

while(buffer\_is\_full(2))

pthread\_cond\_wait(&wait\_empty\_buffer2, &mutex2);

put\_item(item, 2);

printf("\033[33m compute item: %c\n\033[0m", item); //黄色为计算者

pthread\_cond\_signal(&wait\_full\_buffer2);

pthread\_mutex\_unlock(&mutex2);

}

return NULL;

}

void \*produce(void \*arg)

{

char item;

for(int i = 0; i < ITEM\_COUNT; i++)

{

pthread\_mutex\_lock(&mutex1);

while(buffer\_is\_full(1))

pthread\_cond\_wait(&wait\_empty\_buffer1, &mutex1);

item = 'a' + i;

put\_item(item, 1);

printf("\033[31m produce item: %c\n\033[0m", item); //红色为生产者

pthread\_cond\_signal(&wait\_full\_buffer1);

pthread\_mutex\_unlock(&mutex1);

}

return NULL;

}

int main()

{

pthread\_t producer\_tid, computer\_tid, consumer\_tid;

pthread\_mutex\_init(&mutex1, NULL);

pthread\_mutex\_init(&mutex2, NULL);

pthread\_cond\_init(&wait\_empty\_buffer1, NULL);

pthread\_cond\_init(&wait\_empty\_buffer2, NULL);

pthread\_cond\_init(&wait\_full\_buffer1, NULL);

pthread\_cond\_init(&wait\_full\_buffer2, NULL);

pthread\_create(&producer\_tid, NULL, produce, NULL);

pthread\_create(&computer\_tid, NULL, compute, NULL);

pthread\_create(&consumer\_tid, NULL, consume, NULL);

pthread\_join(producer\_tid, NULL);

pthread\_join(computer\_tid, NULL);

pthread\_join(consumer\_tid, NULL);

return 0;

}

## pc2.c: 使用信号量解决生产者、计算者、消费者问题

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#define CAPACITY 4

#define ITEM\_COUNT (CAPACITY \* 2)

char buffer1[CAPACITY];

char buffer2[CAPACITY];

int in1, in2;

int out1, out2;

char get\_item(int n)

{

char item;

if(n == 1)

{

item = buffer1[out1];

out1 = (out1 + 1) % CAPACITY;

}

else if(n == 2)

{

item = buffer2[out2];

out2 = (out2 + 1) % CAPACITY;

}

else

exit(-1);

return item;

}

void put\_item(char item, int n)

{

if(n == 1)

{

buffer1[in1] = item;

in1 = (in1 + 1) % CAPACITY;

}

else if(n == 2)

{

buffer2[in2] = item;

in2 = (in2 + 1) % CAPACITY;

}

else

exit(-1);

}

typedef struct {

int value;

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

}sema\_t;

void sema\_init(sema\_t \*sema, int value)

{

sema->value = value;

pthread\_mutex\_init(&sema->mutex, NULL);

pthread\_cond\_init(&sema->cond, NULL);

}

void sema\_wait(sema\_t \*sema)

{

pthread\_mutex\_lock(&sema->mutex);

int i = 1;

while(sema->value <= 0)

{

pthread\_cond\_wait(&sema->cond, &sema->mutex);

}

sema->value--;

pthread\_mutex\_unlock(&sema->mutex);

}

void sema\_signal(sema\_t \*sema)

{

pthread\_mutex\_lock(&sema->mutex);

sema->value += 1;

pthread\_cond\_signal(&sema->cond);

pthread\_mutex\_unlock(&sema->mutex);

}

sema\_t mutex\_sema1, mutex\_sema2;

sema\_t empty\_buffer\_sema1, empty\_buffer\_sema2;

sema\_t full\_buffer\_sema1, full\_buffer\_sema2;

void \*consume(void \*arg)

{

int item;

for(int i = 0; i < ITEM\_COUNT; i++)

{

sema\_wait(&full\_buffer\_sema2);

sema\_wait(&mutex\_sema2);

item = get\_item(2);

printf("\033[34m consume item: %c\n\033[0m", item); //蓝色为消费者

sema\_signal(&mutex\_sema2);

sema\_signal(&empty\_buffer\_sema2);

}

return NULL;

}

void \*compute(void \*arg)

{

char item;

for(int i = 0; i < ITEM\_COUNT; i++)

{

sema\_wait(&full\_buffer\_sema1);

sema\_wait(&mutex\_sema1);

item = get\_item(1);

sema\_signal(&mutex\_sema1);

sema\_signal(&empty\_buffer\_sema1);

item += 'A' - 'a';

sema\_wait(&empty\_buffer\_sema2);

sema\_wait(&mutex\_sema2);

put\_item(item, 2);

printf("\033[33m compute item: %c\n\033[0m", item); //黄色为计算者

sema\_signal(&mutex\_sema2);

sema\_signal(&full\_buffer\_sema2);

}

return NULL;

}

void \*produce(void \*arg)

{

char item;

for(int i = 0; i < ITEM\_COUNT; i++)

{

sema\_wait(&empty\_buffer\_sema1);

sema\_wait(&mutex\_sema1);

item = 'a' + i;

put\_item(item, 1);

printf("\033[31m produce item: %c\n\033[0m", item); //红色为生产者

sema\_signal(&mutex\_sema1);

sema\_signal(&full\_buffer\_sema1);

}

return NULL;

}

int main()

{

pthread\_t producer\_tid, computer\_tid, consumer\_tid;

sema\_init(&mutex\_sema1, 1);

sema\_init(&mutex\_sema2, 1);

sema\_init(&empty\_buffer\_sema1, CAPACITY - 1);

sema\_init(&empty\_buffer\_sema2, CAPACITY - 1);

sema\_init(&full\_buffer\_sema1, 0);

sema\_init(&full\_buffer\_sema2, 0);

pthread\_create(&producer\_tid, NULL, produce, NULL);

pthread\_create(&computer\_tid, NULL, compute, NULL);

pthread\_create(&consumer\_tid, NULL, consume, NULL);

pthread\_join(producer\_tid, NULL);

pthread\_join(computer\_tid, NULL);

pthread\_join(consumer\_tid, NULL);

return 0;

}

## ring.c: 创建N个线程，它们构成一个环

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#define N 100

int buff[N];

typedef struct {

int value;

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

} sema\_t;

typedef struct {

int order;

} Param;

void sema\_init(sema\_t \*sema, int value)

{

sema->value = value;

pthread\_mutex\_init(&sema->mutex, NULL);

pthread\_cond\_init(&sema->cond, NULL);

}

void sema\_wait(sema\_t \*sema)

{

pthread\_mutex\_lock(&sema->mutex);

while(sema->value <= 0)

pthread\_cond\_wait(&sema->cond, &sema->mutex);

sema->value--;

pthread\_mutex\_unlock(&sema->mutex);

}

void sema\_signal(sema\_t \*sema)

{

pthread\_mutex\_lock(&sema->mutex);

++sema->value;

pthread\_cond\_signal(&sema->cond);

pthread\_mutex\_unlock(&sema->mutex);

}

sema\_t mutex\_sema[N];

sema\_t full\_buffer\_sema[N];

void \*add(void \*arg)

{

int receive;

Param \*param = (Param \*)arg;

int order = param->order;

if(order == 0)

{

sema\_wait(&mutex\_sema[order + 1]);

buff[order + 1] = 1;

sema\_signal(&mutex\_sema[order + 1]);

sema\_signal(&full\_buffer\_sema[order + 1]);

sema\_wait(&full\_buffer\_sema[order]);

sema\_wait(&mutex\_sema[order]);

receive = buff[order];

printf("Thread %d received: %d\n", order + 1, receive);

sema\_signal(&mutex\_sema[order]);

}

else if(order == N - 1)

{

sema\_wait(&full\_buffer\_sema[order]);

sema\_wait(&mutex\_sema[order]);

receive = buff[order];

printf("Thread %d received: %d\n", order + 1, receive);

sema\_signal(&mutex\_sema[order]);

sema\_wait(&mutex\_sema[0]);

buff[0] = receive + 1;

sema\_signal(&mutex\_sema[0]);

sema\_signal(&full\_buffer\_sema[0]);

}

else

{

sema\_wait(&full\_buffer\_sema[order]);

sema\_wait(&mutex\_sema[order]);

receive = buff[order];

printf("Thread %d received: %d\n", order + 1, receive);

sema\_signal(&mutex\_sema[order]);

sema\_wait(&mutex\_sema[order + 1]);

buff[order + 1] = receive + 1;

sema\_signal(&mutex\_sema[order + 1]);

sema\_signal(&full\_buffer\_sema[order + 1]);

}

}

int main()

{

pthread\_t ring\_tid[N];

Param params[N];

for(int i = 0; i < N; i++)

{

sema\_init(&mutex\_sema[i], 1);

sema\_init(&full\_buffer\_sema[i], 0);

}

for(int i = 0; i < N; i++)

{

params[i].order = i;

pthread\_create(&ring\_tid[i], NULL, add, &params[i]);

}

for(int i = 0; i < N; i++)

pthread\_join(ring\_tid[i], NULL);

return 0;

}

## // 题目1

// 主进程创建1个子进程

// 主进程通过管道与子进程连接

// 子进程的标准输出连接到管道的写端

// 主进程的标准输入连接到管道的读端

// 在子进程中调用exec(“echo”, “echo”, “hello world”, NULL)

// 在父进程中调用read(0, buf, sizeof(buf))，从标准输入中获取子进程发送的字符串，并打印出来

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <sys/stat.h>

#include <fcntl.h>

int main()

{

int fd[2];

pipe(fd);

pid\_t pid;

pid = fork();

if(pid == 0)

{

dup2(fd[1], 1);

close(fd[0]);

close(fd[1]);

execlp("echo", "echo", "hello world", NULL);

printf("child process exec failed.\n");

}

else

{

dup2(fd[0], 0);

close(fd[0]);

close(fd[1]);

char buf[1024];

int readsize = read(0, buf, sizeof(buf));

write(1, buf, readsize);

}

wait(NULL);

return 0;

}

## // 题目2

// 主进程创建2个子进程，主进程通过两个管道分别与两个子进程连接

// 第一个子进程计算从1加到50的和，并将结果通过管道送给父进程

// 第一个子进程计算从50加到100的和，并将结果通过管道送给父进程

// 父进程读取两个子进程的结果，将他们相加，打印出来，结果为5050

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <sys/stat.h>

#include <fcntl.h>

int main()

{

int fd1[2];

int fd2[2];

pipe(fd1);

pipe(fd2);

pid\_t pid1;

pid1 = fork();

if(pid1 == 0)

{

close(fd1[0]);

int sum = 0;

for(int i = 1; i <= 50; i++)

sum += i;

write(fd1[1], &sum, sizeof(int));

exit(-1);

}

pid\_t pid2;

pid2 = fork();

if(pid2 == 0)

{

close(fd2[0]);

int sum = 0;

for(int i = 51; i <= 100; i++)

sum += i;

write(fd2[1], &sum, sizeof(int));

exit(-1);

}

int p1, p2;

close(fd1[1]);

close(fd2[1]);

read(fd1[0], &p1, sizeof(int));

read(fd2[0], &p2, sizeof(int));

printf("%d\n", p1 + p2);

return 0;

}

## // 题目3

// 1.主线程创建10个子线程

// - 第0个子线程计算从01加到10的和

// - 第1个子线程计算从11加到20的和

// - 第2个子线程计算从21加到30的和

// - ...

// - 第9个子线程计算从91加到100的和

// 2. 主线程归并10个子线程的计算结果，最终结果为5050

// 3. 本题必须使用线程参数来完成

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define N 10

#define NR\_TOTAL 100

#define NR\_CPU N

#define NR\_CHILD (NR\_TOTAL / NR\_CPU)

typedef struct param {

int start;

int end;

}Param;

typedef struct result {

float sum;

}Result;

void \*compute(void \*arg)

{

Param \*param = (Param \*)arg;

Result \*result;

int sum = 0;

for(int i = param->start + 1; i < param->end + 1; i++)

sum+= i;

result = malloc(sizeof(Result));

result->sum = sum;

return result;

}

int main()

{

pthread\_t workers[NR\_CPU];

Param params[NR\_CPU];

int total = 0;

for(int i = 0; i < NR\_CPU; i++)

{

Param \*param;

param = &params[i];

param->start = i \* NR\_CHILD;

param->end = (i+1) \* NR\_CHILD;

pthread\_create(&workers[i], NULL, compute, param);

}

for(int i = 0; i < NR\_CPU; i++)

{

Result \*result;

pthread\_join(workers[i], (void\*\*)&result);

total += result->sum;

free(result);

}

printf("Total = %d\n", total);

return 0;

}

## // 题目4

// 主线程创建4个子线程T1、T2、T3、T4，主线程在4个子线程退出后，才退出

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#define N 4

int buff[N];

typedef struct {

int value;

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

} sema\_t;

typedef struct {

int order;

} Param;

void sema\_init(sema\_t \*sema, int value)

{

sema->value = value;

pthread\_mutex\_init(&sema->mutex, NULL);

pthread\_cond\_init(&sema->cond, NULL);

}

void sema\_wait(sema\_t \*sema)

{

pthread\_mutex\_lock(&sema->mutex);

while(sema->value <= 0)

pthread\_cond\_wait(&sema->cond, &sema->mutex);

sema->value--;

pthread\_mutex\_unlock(&sema->mutex);

}

void sema\_signal(sema\_t \*sema)

{

pthread\_mutex\_lock(&sema->mutex);

++sema->value;

pthread\_cond\_signal(&sema->cond);

pthread\_mutex\_unlock(&sema->mutex);

}

sema\_t full\_buffer\_sema[N];

void \*T1\_entry(void \*arg)

{

sleep(2); // 睡眠2秒，不准删除此条语句，否则答题无效

puts("T1");

sema\_signal(&full\_buffer\_sema[0]);

sema\_signal(&full\_buffer\_sema[0]);

return NULL;

}

void \*T2\_entry(void \*arg)

{

sleep(1); // 睡眠1秒，不准删除此条语句，否则答题无效

sema\_wait(&full\_buffer\_sema[0]);

puts("T2");

sema\_signal(&full\_buffer\_sema[1]);

return NULL;

}

void \*T3\_entry(void \*arg)

{

sleep(1); // 睡眠1秒，不准删除此条语句，否则答题无效

sema\_wait(&full\_buffer\_sema[0]);

puts("T3");

sema\_signal(&full\_buffer\_sema[2]);

return NULL;

}

void \*T4\_entry(void \*arg)

{

sema\_wait(&full\_buffer\_sema[1]);

sema\_wait(&full\_buffer\_sema[2]);

puts("T4");

return NULL;

}

int main()

{

pthread\_t tid[4];

for(int i = 0; i < N; i++)

sema\_init(&full\_buffer\_sema[i], 0);

pthread\_create(&tid[0], NULL, T1\_entry, NULL);

pthread\_create(&tid[1], NULL, T2\_entry, NULL);

pthread\_create(&tid[2], NULL, T3\_entry, NULL);

pthread\_create(&tid[3], NULL, T4\_entry, NULL);

for(int i = 0; i < N; i++)

pthread\_join(tid[i], NULL);

return 0;

}

# 文件读写编程题目

## myecho.c

* myecho.c的功能与系统echo程序相同
* 接受命令行参数，并将参数打印出来，例子如下：
* $ ./myecho x
* x
* $ ./myecho a b c
* a b c

## mycat.c

* mycat.c的功能与系统cat程序相同
* mycat将指定的文件内容输出到屏幕，例子如下：
* 要求使用系统调用open/read/write/close实现
* $ cat /etc/passwd
* root:x:0:0:root:/root:/bin/bash
* daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
* bin:x:2:2:bin:/bin:/usr/sbin/nologin
* ...
* $ ./mycat /etc/passwd
* root:x:0:0:root:/root:/bin/bash
* daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
* bin:x:2:2:bin:/bin:/usr/sbin/nologin
* ...

## mycp.c

* mycp.c的功能与系统cp程序相同
* 将源文件复制到目标文件，例子如下：
* 要求使用系统调用open/read/write/close实现
* $ cat /etc/passwd
* root:x:0:0:root:/root:/bin/bash
* daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
* bin:x:2:2:bin:/bin:/usr/sbin/nologin
* ...
* $ ./mycp /etc/passwd passwd.bak
* $ cat passwd.bak
* root:x:0:0:root:/root:/bin/bash
* daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
* bin:x:2:2:bin:/bin:/usr/sbin/nologin
* ...

# 多进程题目

## mysys.c: 实现函数mysys，用于执行一个系统命令，要求如下

* mysys的功能与系统函数system相同，要求用进程管理相关系统调用自己实现一遍
* 使用fork/exec/wait系统调用实现mysys
* 不能通过调用系统函数system实现mysys
* 测试程序
* #include <stdio.h>
* int main()
* {
* printf("--------------------------------------------------\n");
* system("echo HELLO WORLD");
* printf("--------------------------------------------------\n");
* system("ls /");
* printf("--------------------------------------------------\n");
* return 0;
* }
* 测试程序的输出结果
* --------------------------------------------------
* HELLO WORLD
* --------------------------------------------------
* bin core home lib mnt root snap tmp vmlinuz
* boot dev initrd.img lost+found opt run srv usr vmlinuz.old
* cdrom etc initrd.img.old media proc sbin sys var
* --------------------------------------------------

## sh1.c: 实现shell程序，要求具备如下功能

* 支持命令参数
* $ echo arg1 arg2 arg3
* $ ls /bin /usr/bin /home
* 实现内置命令cd、pwd、exit
* $ cd /bin
* $ pwd
* /bin

## sh2.c: 实现shell程序，要求在第1版的基础上，添加如下功能

* 实现文件重定向
* $ echo hello >log
* $ cat log
* hello

## sh3.c: 实现shell程序，要求在第2版的基础上，添加如下功能

* 实现管道
* $ cat /etc/passwd | wc -l
* 实现管道和文件重定向
* $ cat input.txt
* 3
* 2
* 1
* 3
* 2
* 1
* $ cat <input.txt | sort | uniq | cat >output.txt
* $ cat output.txt
* 1
* 2
* 3

# 多线程题目

## pi1.c: 使用2个线程根据莱布尼兹级数计算PI

* 莱布尼兹级数公式: 1 - 1/3 + 1/5 - 1/7 + 1/9 - ... = PI/4
* 主线程创建1个辅助线程
* 主线程计算级数的前半部分
* 辅助线程计算级数的后半部分
* 主线程等待辅助线程运行結束后,将前半部分和后半部分相加

## pi2.c: 使用N个线程根据莱布尼兹级数计算PI

* 与上一题类似，但本题更加通用化，能适应N个核心，需要使用线程参数来实现
* 主线程创建N个辅助线程
* 每个辅助线程计算一部分任务，并将结果返回
* 主线程等待N个辅助线程运行结束，将所有辅助线程的结果累加

## sort.c: 多线程排序

* 主线程创建一个辅助线程
* 主线程使用选择排序算法对数组的前半部分排序
* 辅助线程使用选择排序算法对数组的后半部分排序
* 主线程等待辅助线程运行結束后,使用归并排序算法归并数组的前半部分和后半部分

## pc1.c: 使用条件变量解决生产者、计算者、消费者问题

* 系统中有3个线程：生产者、计算者、消费者
* 系统中有2个容量为4的缓冲区：buffer1、buffer2
* 生产者生产'a'、'b'、'c'、‘d'、'e'、'f'、'g'、'h'八个字符，放入到buffer1
* 计算者从buffer1取出字符，将小写字符转换为大写字符，放入到buffer2
* 消费者从buffer2取出字符，将其打印到屏幕上

## pc2.c: 使用信号量解决生产者、计算者、消费者问题

* 功能和前面的实验相同，使用信号量解决

## ring.c: 创建N个线程，它们构成一个环

* 创建N个线程：T1、T2、T3、… TN
* T1向T2发送整数1
* T2收到后将整数加1
* T2向T3发送整数2
* T3收到后将整数加1
* T3向T4发送整数3
* …
* TN收到后将整数加1
* TN向T1发送整数N