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_CPUN
#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