**《进程同步控制》实验报告**

**一、实验目的**

本实验旨在动手设计一个进程同步控制实验，更深刻的理解进程之间的协作机制

**二、实验内容**

**2.1 实验内容**

* 利用信号量机制，提供读者-写者问题的实现方案，并分别实现读者优先与写者优先。
* 读者-写者问题的读写操作限制：
  + 写-写互斥：不能有两个写者同时进行写操作。
  + 读-写互斥：不能同时有一个线程在读，一个进程在写。
  + 读-读允许：允许多个读者同时执行读操作。

**读者优先：**在实现上述限制的同时，要求读者的操作优先级高于写者。要求没有读者保 持等待除非已有一个写者已经被允许使用共享数据。

**写者优先：**在实现上述限制的同时，要求写者的操作权限高于写者。要求一旦写者就绪， 那么将不会有新的读者开始读操作

**2.2 实验要求**

* 实验环境：在 OpenEuler/Linux 环境下，使用 C/C++开发环境。
* 程序要求：

1. 创建一个包含n个线程的控制台程序，并用这n个线程表示n个读者或写者。

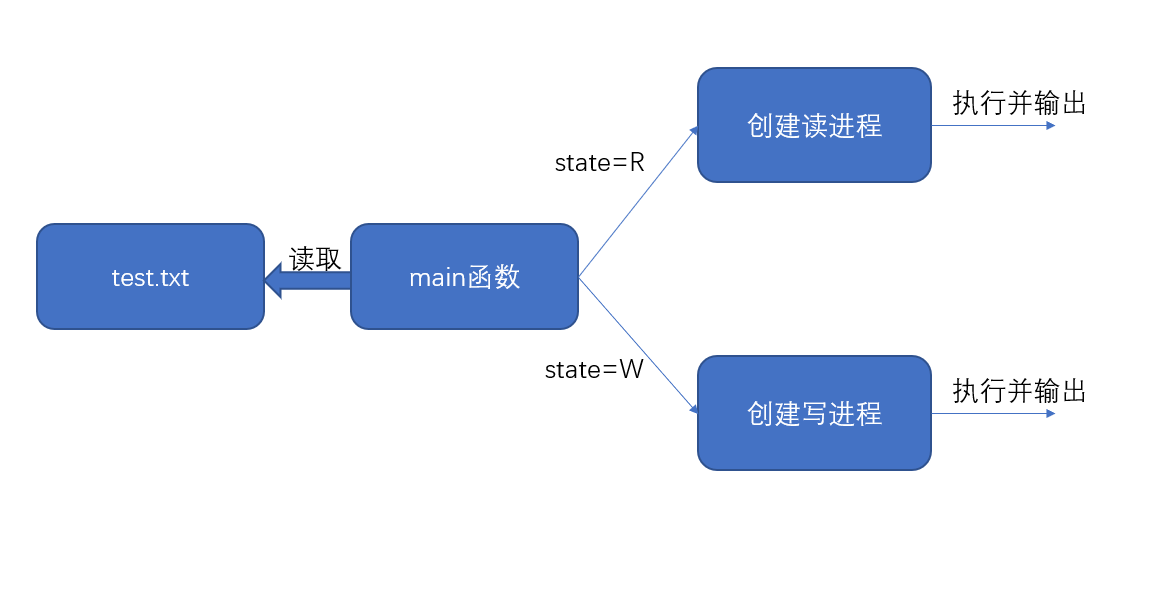
2. 利用信号量机制，分别实现满足读者优先与写者优先的读者-写者问题。

3. 输入要求：要求使用文件输入相应命令，并根据这些命令创建相应的读写进程。

4. 输出要求：要求运行结果在控制台输出并保存在相应文件中。输出内容包括线程创建提示、线程进入共享缓冲区提示、线程操作执行提示、线程离开缓冲区提示。

**三、实验原理**

**3.1 程序流程**



**3.2 文件读入与进程创建**

**1.数据结构定义**

struct TInfo {

int id;//进程id

char operation;//进程操作类型

int start\_time;//进程的开始时间

int last\_time;//进程的持续时间

};//读入文件的每一行储存在一个这样的结构中，代表一个进程

pthread\_t tid;

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);//创建进程的初始化条件

char filename[20];//读入文件名

int lines = atoi(argv[2]);//文件行数

int pthread\_create()//进程创建函数

**2.算法实现**

FILE\* fp = fopen(filename, "r");//打开文件

int i = 0;

for (i = 0; i < lines; i++) {//逐行读取并根据进程类型 创建不同的读写进程

struct TInfo\* t = (struct TInfo\*)malloc(sizeof(struct TInfo));

fscanf(fp, "%d %c %d %d\n", &t->id, &t->operation, &t->start\_time, &t->last\_time);

if (t->operation == 'R') {//创建读进程

pthread\_create(&tid, &attr, RP\_ReaderThread, t);

}

else if (t->operation == 'W') {//创建写进程

pthread\_create(&tid, &attr, RP\_WriterThread, t);

}

}

**3.3 读者优先逻辑**

**1.数据结构定义**

int id = ((struct TInfo\*)args)->id;//进程id

int start\_time = ((struct TInfo\*)args)->start\_time;//进程开始时间

int last\_time = ((struct TInfo\*)args)->last\_time;//进程持续时间

sem\_t RP\_Write, mutex;//控制写进程的锁 以及保护read\_count的锁

int read\_count;//当前读者人数

vector<int> buffer;//缓冲区

int numMax = 1000;//max\_随机数

clock\_t start;//程序被创立时间

**2.读者进程函数**

通过Sleep(start\_time)可以实现让进程一直等待，一直到进程开始时间时才开始运行

通过Sleep(last\_time)达到进程持续运行XX长时间的目的

void\* RP\_ReaderThread(void\* args) {

int id = ((struct TInfo\*)args)->id;

int start\_time = ((struct TInfo\*)args)->start\_time;

int last\_time = ((struct TInfo\*)args)->last\_time;

clock\_t t;

sleep(start\_time);//通过sleep达到在开始时间开始的目的

printf("ReaderThread %d: waiting to read\n", id);

sem\_wait(&mutex);//read\_count线程锁

read\_count++;

if (read\_count == 1)

sem\_wait(&RP\_Write);//写锁关闭

sem\_post(&mutex);

printf("ReaderThread %d: start reading at ", id);

t = clock();

cout << t - start << endl;

sleep(last\_time);

if (buffer.size() != 0)

{

int index = rand() % buffer.size();

int item = buffer[index];

printf("Reader Thread:%d Read %d from buffer\n", id, item);

}

else {

printf("buffer is empty!\n");

}

printf("ReaderThread %d: end reading at ", id);

t = clock();

cout << t - start << endl;

sem\_wait(&mutex);

read\_count--;

if (read\_count == 0)

sem\_post(&RP\_Write);

sem\_post(&mutex);

pthread\_exit(0);

}

读者优先的逻辑下，读者优先的设计思想是读进程只要看到有其它读进程正在读，就可以继续进行读；该算法只要还有一个读者在活动，就允许后续的读者进来，该策略的结果是，如果有一个稳定的读者流存在，那么这些读者将在到达后被允许进入。

**3.写者进程函数**

void\* RP\_WriterThread(void\* args) {

int id = ((struct TInfo\*)args)->id;

int start\_time = ((struct TInfo\*)args)->start\_time;

int last\_time = ((struct TInfo\*)args)->last\_time;

int item;

clock\_t t;

sleep(start\_time);//开始时间

printf("WriterThread %d: waiting to write\n", id);

sem\_wait(&RP\_Write);

printf("WriterThread %d: start writing at ", id);

t = clock();

cout << t - start << endl;

sleep(last\_time);

item = rand() % numMax;

buffer.push\_back(item);

printf("WriteThread:%d Write in %d\n", id, item);//随机生成数写入缓冲区

printf("WriterThread %d: end writing at ", id);

t = clock();

cout << t - start << endl;

sem\_post(&RP\_Write);

pthread\_exit(0);

}

写进程必须等待所有读进程都不读时才能写，即使写进程可能比一些读进程更早提出申请。写者在只要有一个读者进程在运行时，就始终被挂起，直到没有读者为止。

**3.4写者优先逻辑**

**1.数据结构定义**

int id = ((struct TInfo\*)args)->id;//进程id

int start\_time = ((struct TInfo\*)args)->start\_time;//进程开始时间

int last\_time = ((struct TInfo\*)args)->last\_time;//进程持续时间

sem\_t cs\_Write, mutex1, mutex2, cs\_Read;//控制写进程的锁，read\_count和write\_count保护锁，控制读进程的锁

int read\_count, write\_count;//当前读者人数和写者人数vector<int> buffer;//缓冲区

int numMax = 1000;//max\_随机数

clock\_t start;//程序被创立时间

**2.读者进程函数**

void\* WP\_ReaderThread(void\* args) {

int id = ((struct TInfo\*)args)->id;

int start\_time = ((struct TInfo\*)args)->start\_time;

int last\_time = ((struct TInfo\*)args)->last\_time;

clock\_t t;

sleep(start\_time);

printf("ReaderThread %d: waiting to read\n", id);

sem\_wait(&cs\_Read);//排队信号量，读进程每次操作前需要等待该信号量

sem\_wait(&mutex2);//read\_count保护

read\_count++;

if (read\_count == 1)

sem\_wait(&cs\_Write);

sem\_post(&mutex2);

sem\_post(&cs\_Read);//释放

printf("ReaderThread %d: start reading at ", id);

t = clock();

cout << t - start << endl;

sleep(last\_time);

if (buffer.size() != 0)

{

int index = rand() % buffer.size();

int item = buffer[index];

printf("Reader Thread:%d Read %d from buffer\n", id, item);

}

else {

printf("buffer is empty!\n");

}

printf("ReaderThread %d: end reading at ", id);

t = clock();

cout << t - start << endl;

sem\_wait(&mutex2);

read\_count--;

if (read\_count == 0)

sem\_post(&cs\_Write);

sem\_post(&mutex2);

pthread\_exit(0);

}

在读者优先的算法的基础上增加了一个排队信号量 cs\_Read，读、写进程在每次

操作前都要等待 cs\_Read 信号量。

**3.写者进程函数**

void\* WP\_WriterThread(void\* args) {

int id = ((struct TInfo\*)args)->id;

int start\_time = ((struct TInfo\*)args)->start\_time;

int last\_time = ((struct TInfo\*)args)->last\_time;

int item;

clock\_t t;

sleep(start\_time);

printf("WriterThread %d: waiting to write\n", id);

sem\_wait(&mutex1);

write\_count++;

if (write\_count == 1)

sem\_wait(&cs\_Read);//排队锁锁住

sem\_post(&mutex1);

sem\_wait(&cs\_Write);

printf("WriterThread %d: start writing at ", id);

t = clock();

cout << t - start << endl;

sleep(last\_time);

item = rand() % numMax;

buffer.push\_back(item);

printf("WriteThread:%d Write in %d\n", id, item);//随机生成数写入缓冲区

printf("WriterThread %d: end writing at ", id);

t = clock();

cout << t - start << endl;

sem\_post(&cs\_Write);

sem\_wait(&mutex1);

write\_count--;

if (write\_count == 0)

sem\_post(&cs\_Read);

sem\_post(&mutex1);

pthread\_exit(0);

}

写者优先的设计思想是在一个写者到达时如果有正在工作的读者，那么该写

者只要等待正在工作的读者完成，而不必等候其后面到来的读者就可以进行写操

作。该算法当一个写者在等待时，后到达的读者是在写者之后被挂起，而不是立

即允许进入。

**四、实验环境**

*  操作系统：Ubuntu 16.04 LTS
*  编译环境：g++编译器

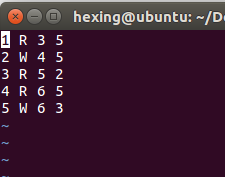
**五、实验步骤**

* **读者优先**

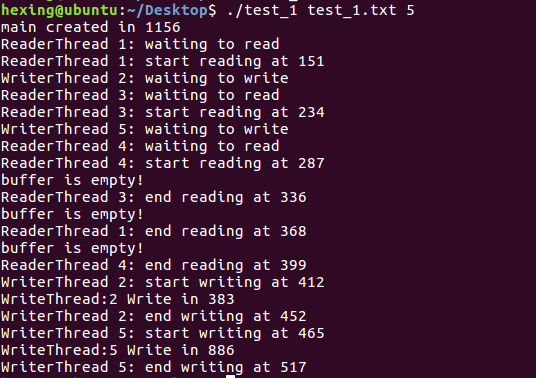
**1.编译生成可执行程序**



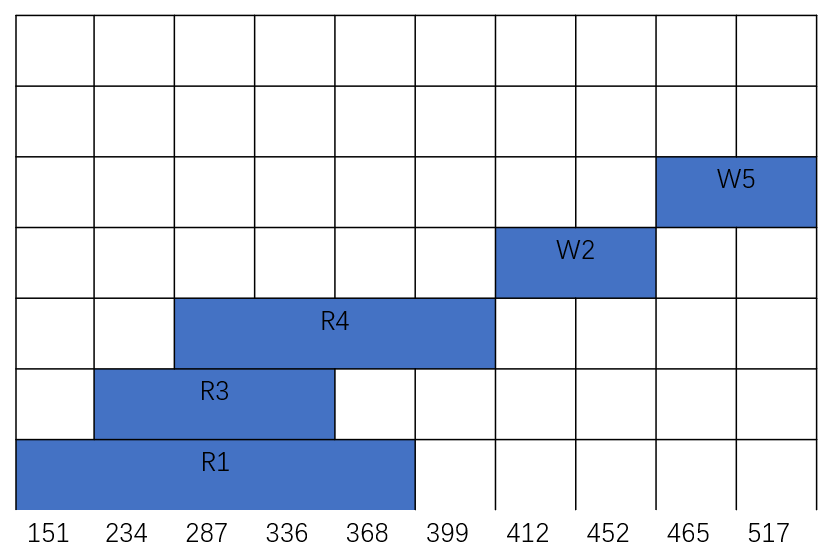
**2.编写test.txt测试**



**3.执行程序**



由执行结果，可以绘制如下的甘特图



通过甘特图我们可以看出，第一个进程为读者进程，第二个进程为写者进程，但写者进程W2一直等到所有读者进程都执行完之后才开始执行，满足读者优先的策略

* **写者优先**

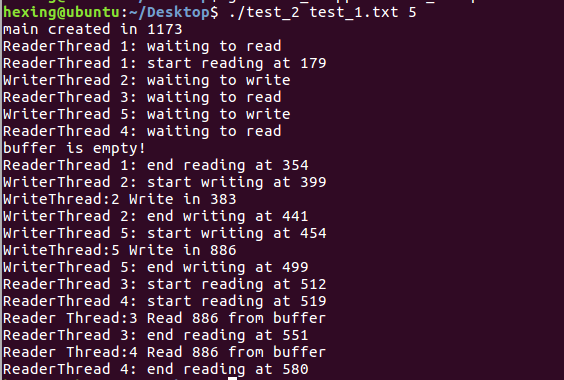
**1.编译生成可执行程序**



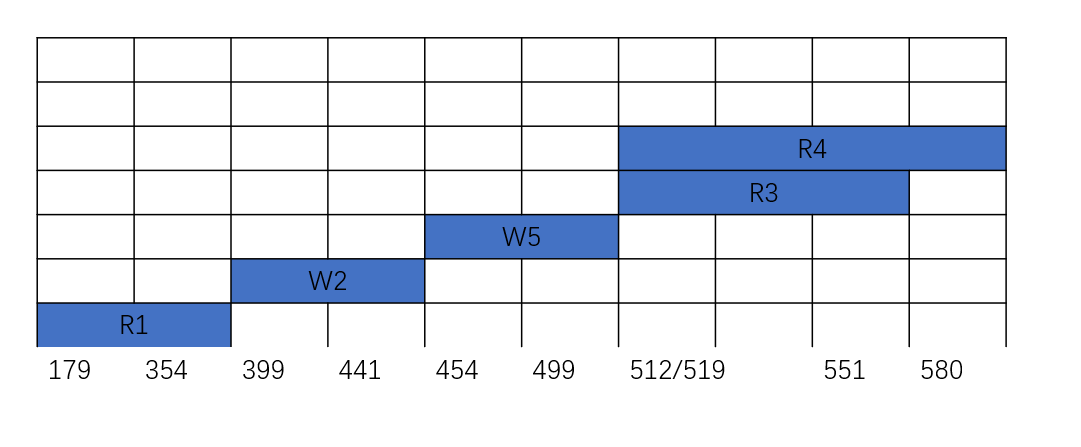
**2.编写test.txt测试**

这里与读者优先的文件相同，起到对比的作用

**3.执行程序**



根据执行结果，可以画出如下的甘特图：



甘特图的展示效果与期望中的相同，当读者进程1执行完毕时，写者进程2此时执行，后到达的读者进程，会等待所有写者进程执行完毕后执行，符合写者优先的策略

**附录：**

* **读者优先**
* //Reader
* #include<stdio.h>
* #include<stdlib.h>
* #include<sys/types.h>
* #include<pthread.h>
* #include<string.h>
* #include<semaphore.h>
* #include<unistd.h>
* #include<vector>
* #include<time.h>
* #include<iostream>
* **using** **namespace** std;
* //semaphores
* sem\_t RP\_Write,mutex;
* **int** read\_count;
* vector<**int**> buffer;
* **int** numMax = 1000;//max\_suijishu
* **clock\_t** start;
* **struct** TInfo{
* **int** id;
* **char** operation;
* **int** start\_time;
* **int** last\_time;
* };
* **void** \*RP\_ReaderThread(**void** \*args){
* **int** id=((**struct** TInfo\*)args)->id;
* **int** start\_time=((**struct** TInfo\*)args)->start\_time;
* **int** last\_time=((**struct** TInfo\*)args)->last\_time;
* **clock\_t** t;
* sleep(start\_time);
* printf("ReaderThread %d: waiting to read\n",id);
* sem\_wait(&mutex);
* read\_count++;
* **if**(read\_count==1)
* sem\_wait(&RP\_Write);
* sem\_post(&mutex);
* printf("ReaderThread %d: start reading at ",id);
* t=clock();
* cout<<t-start<<endl;
* sleep(last\_time);
* **if**(buffer.size()!=0)
* {
* **int** index = rand()%buffer.size();
* **int** item = buffer[index];
* printf("Reader Thread:%d Read %d from buffer\n",id,item);
* }
* **else**{
* printf("buffer is empty!\n");
* }
* printf("ReaderThread %d: end reading at ",id);
* t=clock();
* cout<<t-start<<endl;
* sem\_wait(&mutex);
* read\_count--;
* **if**(read\_count==0)
* sem\_post(&RP\_Write);
* sem\_post(&mutex);
* pthread\_exit(0);
* }
* **void** \*RP\_WriterThread(**void** \*args){
* **int** id=((**struct** TInfo\*)args)->id;
* **int** start\_time=((**struct** TInfo\*)args)->start\_time;
* **int** last\_time=((**struct** TInfo\*)args)->last\_time;
* **int** item;
* **clock\_t** t;
* sleep(start\_time);
* printf("WriterThread %d: waiting to write\n",id);
* sem\_wait(&RP\_Write);
* printf("WriterThread %d: start writing at ",id);
* t=clock();
* cout<<t-start<<endl;
* sleep(last\_time);
* item = rand()%numMax;
* buffer.push\_back(item);
* printf("WriteThread:%d Write in %d\n",id,item);
* printf("WriterThread %d: end writing at ",id);
* t=clock();
* cout<<t-start<<endl;
* sem\_post(&RP\_Write);
* pthread\_exit(0);
* }
* **int** main(**int** argc,**char** \*argv[]){
* pthread\_t tid;
* pthread\_attr\_t attr;
* pthread\_attr\_init(&attr);
* **char** filename[20];
* **int** lines=atoi(argv[2]);
* strcpy(filename,argv[1]);
* sem\_init(&mutex,0,1);
* sem\_init(&RP\_Write,0,1);
* read\_count=0;
* start=clock();
* cout<<"main created in "<<start<<endl;
* **FILE** \*fp=fopen(filename,"r");
* **int** i=0;
* **for**(i=0;i<lines;i++){
* **struct** TInfo\* t = (**struct** TInfo\*)malloc(**sizeof**(**struct** TInfo));
* fscanf(fp,"%d %c %d %d\n",&t->id,&t->operation,&t->start\_time,&t->last\_time);
* **if**(t->operation == 'R'){
* pthread\_create(&tid,&attr,RP\_ReaderThread,t);
* }
* **else** **if**(t->operation == 'W'){
* pthread\_create(&tid,&attr,RP\_WriterThread,t);
* }
* }
* sleep(40);
* **return** 0;
* }
* **写者优先**
* //Write
* #include<stdio.h>
* #include<stdlib.h>
* #include<sys/types.h>
* #include<pthread.h>
* #include<string.h>
* #include<semaphore.h>
* #include<unistd.h>
* #include<vector>
* #include<time.h>
* #include<iostream>
* **using** **namespace** std;
* //semaphores
* sem\_t cs\_Write,mutex1,mutex2,cs\_Read;
* **int** read\_count,write\_count;
* vector<**int**> buffer;
* **int** numMax = 1000;//max\_suijishu
* **clock\_t** start;
* **struct** TInfo{
* **int** id;
* **char** operation;
* **int** start\_time;
* **int** last\_time;
* };
* **void** \*WP\_ReaderThread(**void** \*args){
* **int** id=((**struct** TInfo\*)args)->id;
* **int** start\_time=((**struct** TInfo\*)args)->start\_time;
* **int** last\_time=((**struct** TInfo\*)args)->last\_time;
* **clock\_t** t;
* sleep(start\_time);
* printf("ReaderThread %d: waiting to read\n",id);
* sem\_wait(&cs\_Read);
* sem\_wait(&mutex2);
* read\_count++;
* **if**(read\_count==1)
* sem\_wait(&cs\_Write);
* sem\_post(&mutex2);
* sem\_post(&cs\_Read);
* printf("ReaderThread %d: start reading at ",id);
* t = clock();
* cout << t - start << endl;
* sleep(last\_time);
* **if** (buffer.size() != 0)
* {
* **int** index = rand() % buffer.size();
* **int** item = buffer[index];
* printf("Reader Thread:%d Read %d from buffer\n", id, item);
* }
* **else** {
* printf("buffer is empty!\n");
* }
* printf("ReaderThread %d: end reading at ",id);
* t = clock();
* cout << t - start << endl;
* sem\_wait(&mutex2);
* read\_count--;
* **if**(read\_count==0)
* sem\_post(&cs\_Write);
* sem\_post(&mutex2);
* pthread\_exit(0);
* }
* **void** \*WP\_WriterThread(**void** \*args){
* **int** id=((**struct** TInfo\*)args)->id;
* **int** start\_time=((**struct** TInfo\*)args)->start\_time;
* **int** last\_time=((**struct** TInfo\*)args)->last\_time;
* **int** item;
* **clock\_t** t;
* sleep(start\_time);
* printf("WriterThread %d: waiting to write\n",id);
* sem\_wait(&mutex1);
* write\_count++;
* **if**(write\_count==1)
* sem\_wait(&cs\_Read);
* sem\_post(&mutex1);
* sem\_wait(&cs\_Write);
* printf("WriterThread %d: start writing at ",id);
* t = clock();
* cout << t - start << endl;
* sleep(last\_time);
* item = rand() % numMax;
* buffer.push\_back(item);
* printf("WriteThread:%d Write in %d\n", id, item);//随机生成数写入缓冲区
* printf("WriterThread %d: end writing at ",id);
* t = clock();
* cout << t - start << endl;
* sem\_post(&cs\_Write);
* sem\_wait(&mutex1);
* write\_count--;
* **if**(write\_count==0)
* sem\_post(&cs\_Read);
* sem\_post(&mutex1);
* pthread\_exit(0);
* }
* **int** main(**int** argc,**char** \*argv[]){
* pthread\_t tid;
* pthread\_attr\_t attr;
* pthread\_attr\_init(&attr);
* **char** filename[20];
* **int** lines=atoi(argv[2]);
* strcpy(filename,argv[1]);
* sem\_init(&mutex1,0,1);
* sem\_init(&mutex2,0,1);
* sem\_init(&cs\_Write,0,1);
* sem\_init(&cs\_Read,0,1);
* read\_count=0;
* write\_count=0;
* start = clock();
* cout << "main created in " << start << endl;
* **FILE** \*fp=fopen(filename,"r");
* **int** i=0;
* **for**(i=0;i<lines;i++){
* **struct** TInfo\* t = (**struct** TInfo\*)malloc(**sizeof**(**struct** TInfo));
* fscanf(fp,"%d %c %d %d\n",&t->id,&t->operation,&t->start\_time,&t->last\_time);
* **if**(t->operation == 'R'){
* pthread\_create(&tid,&attr,WP\_ReaderThread,t);
* }
* **else** **if**(t->operation == 'W'){
* pthread\_create(&tid,&attr,WP\_WriterThread,t);
* }
* }
* sleep(40);
* **return** 0;
* }