**源代码**

#include <windows.h>

#include <ctype.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <malloc.h>

#define MAX\_PERSON 100

#define READER 0 //读者

#define WRITER 1 //写者

#define END -1

#define R READER

#define W WRITER

typedef struct \_Person

{

HANDLE m\_hThread;//定义处理线程的句柄

int m\_nType;//进程类型（读写）

int m\_nStartTime;//开始时间

int m\_nWorkTime;//运行时间

int m\_nID;//进程号

}Person;

Person g\_Persons[MAX\_PERSON];

int g\_NumPerson = 0;

long g\_CurrentTime= 0;//基本时间片数

int g\_PersonLists[] = {//进程队列

1, W, 4, 5, 2, W, 16, 4, 3, R, 5, 2,

4, W, 6, 5, 5, R, 4, 3,

END,

};

int g\_NumOfReading = 0;

int g\_NumOfWriteRequest = 0;//申请写进程的个数

HANDLE g\_hReadSemaphore;//读者信号

HANDLE g\_hWriteSemaphore;//写者信号

bool finished = false; //所有的读完成

//bool wfinished = false; //所有的写完成

void CreatePersonList(int \*pPersonList);

bool CreateReader(int StartTime,int WorkTime,int ID);

bool CreateWriter(int StartTime,int WorkTime,int ID);

DWORD WINAPI ReaderProc(LPVOID lpParam);

DWORD WINAPI WriterProc(LPVOID lpParam);

int main()

{

g\_hReadSemaphore = CreateSemaphore(NULL,1,100,NULL); //创建信号灯，当前可用的资源数为，最大为

g\_hWriteSemaphore = CreateSemaphore(NULL,1,100,NULL); //创建信号灯，当前可用的资源数为，最大为

CreatePersonList(g\_PersonLists); // Create All the reader and writers

printf("Created all the reader and writer\n...\n");

g\_CurrentTime = 0;

while(true)

{

g\_CurrentTime++;

Sleep(300); // 300 ms

printf("CurrentTime = %d\n",g\_CurrentTime);

if(finished) return 0;

system ("pause");

} // return 0;

}

void CreatePersonList(int \*pPersonLists)

{

int i=0;

int \*pList = pPersonLists;

bool Ret;

while(pList[0] != END)

{

switch(pList[1])

{

case R:

Ret = CreateReader(pList[2],pList[3],pList[0]);//351,w452,523,654

break; case W:

Ret = CreateWriter(pList[2],pList[3],pList[0]);

break;

}

if(!Ret)

printf("Create Person %d is wrong\n",pList[0]);

pList += 4; // move to next person list

}

}

DWORD WINAPI ReaderProc(LPVOID lpParam)//读过程

{

Person \*pPerson = (Person\*)lpParam;

// wait for the start time

while(g\_CurrentTime != pPerson->m\_nStartTime)

{ }

printf("Reader %d is Requesting ...\n",pPerson->m\_nID);

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

// wait for the write request

WaitForSingleObject(g\_hReadSemaphore,INFINITE); if(g\_NumOfReading ==0)

{

WaitForSingleObject(g\_hWriteSemaphore,INFINITE); }

g\_NumOfReading++;

ReleaseSemaphore(g\_hReadSemaphore,1,NULL);

pPerson->m\_nStartTime = g\_CurrentTime;

printf("Reader %d is Reading the Shared Buffer...\n",pPerson->m\_nID);

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

while(g\_CurrentTime <= pPerson->m\_nStartTime + pPerson->m\_nWorkTime)

{}

printf("Reader %d is Exit...\n",pPerson->m\_nID);

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

WaitForSingleObject(g\_hReadSemaphore,INFINITE);

g\_NumOfReading--;

if(g\_NumOfReading == 0)

{ReleaseSemaphore(g\_hWriteSemaphore,1,NULL);//此时没有读者，可以写

}

ReleaseSemaphore(g\_hReadSemaphore,1,NULL);

if(pPerson->m\_nID == 4) finished = true; //所有的读写完成

ExitThread(0);

return 0;

}

DWORD WINAPI WriterProc(LPVOID lpParam)

{

Person \*pPerson = (Person\*)lpParam;

// wait for the start time

while(g\_CurrentTime != pPerson->m\_nStartTime)

{}

printf("Writer %d is Requesting ...\n",pPerson->m\_nID);

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

WaitForSingleObject(g\_hWriteSemaphore,INFINITE);

// modify the writer's real start time

pPerson->m\_nStartTime = g\_CurrentTime;

printf("Writer %d is Writting the Shared Buffer...\n",pPerson->m\_nID);

while(g\_CurrentTime <= pPerson->m\_nStartTime + pPerson->m\_nWorkTime)

{}

printf("Writer %d is Exit...\n",pPerson->m\_nID);

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

//g\_NumOfWriteRequest--;

ReleaseSemaphore(g\_hWriteSemaphore,1,NULL);

if(pPerson->m\_nID == 4) finished = true;//所有的读写完成

ExitThread(0);

return 0;

}

bool CreateReader(int StartTime,int WorkTime,int ID)

{

DWORD dwThreadID;

if(g\_NumPerson >= MAX\_PERSON)

return false;

Person \*pPerson = &g\_Persons[g\_NumPerson];

pPerson->m\_nID = ID;

pPerson->m\_nStartTime = StartTime;

pPerson->m\_nWorkTime = WorkTime;

pPerson->m\_nType = READER;

g\_NumPerson++;

// Create an New Thread

pPerson->m\_hThread= CreateThread(NULL,0,ReaderProc,(LPVOID)pPerson,0,&dwThreadID);

if(pPerson->m\_hThread == NULL)

return false;

return true;

}

bool CreateWriter(int StartTime,int WorkTime,int ID)

{

DWORD dwThreadID;

if(g\_NumPerson >= MAX\_PERSON)

return false;

Person \*pPerson = &g\_Persons[g\_NumPerson];

pPerson->m\_nID = ID;

pPerson->m\_nStartTime = StartTime;

pPerson->m\_nWorkTime = WorkTime;

pPerson->m\_nType = WRITER;

g\_NumPerson++;

// Create an New Thread

pPerson->m\_hThread= CreateThread(NULL,0,WriterProc,(LPVOID)pPerson,0,&dwThreadID);

if(pPerson->m\_hThread == NULL)

return false;

return true;

}