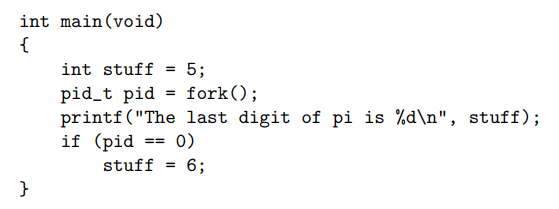
**作业一**

一、简答题

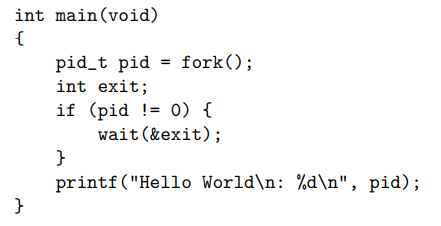
1. 执行以下函数会输出什么？



The last digit of pi is 5

The last digit of pi is 5

1. 执行以下函数会输出什么？Assume the child PID is 90210.



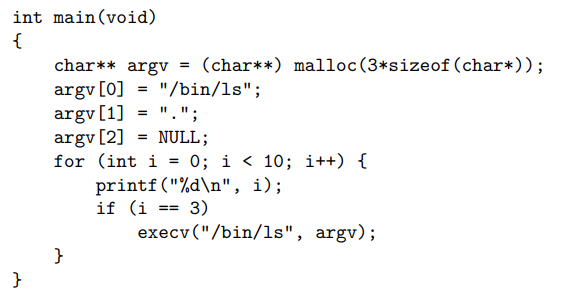
Hello World

: 0

Hello World

: 90210

1. 执行以下函数会输出什么？



0

1

2

3

<output of ls>

1. 考虑以下伪代码段：

pid t pid;

pid = fork();

if (pid == 0) { /\* child process \*/

fork();

thread create( . . .);

}

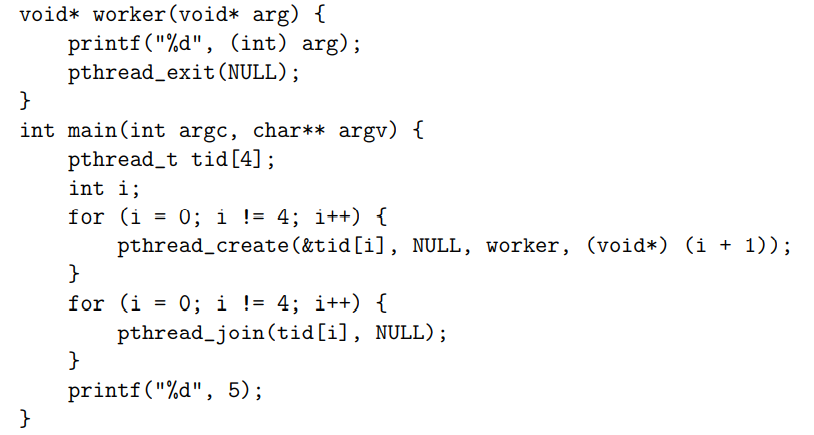
fork();

多少个进程被创建？ 多少个线程被创建？

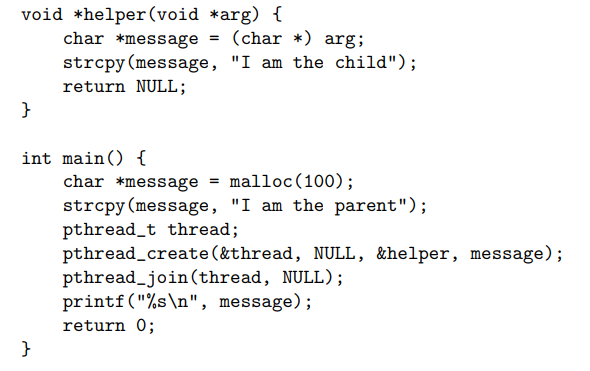
5个进程

2个线程

1. 假设pthread的库函数总能被成功的调用，不出错。对以下代码，下面哪句描述是错误的？ **B**



1. 程序运行的结果可以是21345
2. 程序运行的结果可以是1235
3. 被创建的4个worker线程退出的顺序可以和它们被创建的顺序是一致的。
4. 被创建的4个worker线程退出的顺序也可以和它们被创建的顺序是不一致的。
5. 当执行到main函数中的printf 语句，所有四个worker线程一定执行过了pthread\_exit
6. 给出程序运行的输出结果：



I am the child

二、 综合问答题

1、下表给出几个进程的到达时间和所需的运行时间。

**Process Arrival time Burst time**

**P1 0 17**

**P2 12 25**

**P3 28 8**

**P4 36 32**

**P5 46 18**

1. 用甘特图（Gantt Chart）描述进程在多级反馈队列调度 (Multilevel Feedback Queue Scheduling) 下的执行顺序。使用三级队列，满足：queue 0: quantum 8, queue 1: quantum 16, queue 2: FCFS。
2. 计算进程上下文切换次数。
3. 计算平均等待时间和平均周转时间。

(1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P1 | P2 | P1 | P2 | P3 | P4 | P2 | P5 | P4 | P2 | P5 | P4 |

0 8 12 20 25 28 36 44 46 54 70 82 92 100

(2)上下文切换次数为8次，上图中标红的为上下文切换处，其他位置为进程结束时主动让出处理器不再需要保留状态，不是上下文切换。

(3)平均等待时间

平均周转时间

2、请考虑调度以下四个进程，

**Process Arrival Time Burst Time**

**P1 0 5**

**P2 3 6**

**P3 6 4**

**P4 9 7**

1. Draw the Gantt chart and calculate the average waiting time & average response time for nonpreemptive FCFS algorithm (no time quantum). 在没有时间片的情况下，请画出先来先服务调度算法的甘特图、算出其平均等待时间和平均响应时间。
2. Draw the Gantt chart and calculate the average waiting time & average response time for SRTF (or called “shortest-remaining-time-first”) algorithm (no time quantum).

在没有时间片的情况下，请画出最短剩余时间优先调度算法的甘特图、算出其平均等待时间和平均响应时间

1. If the priorities of P1, P2, P3, P4 are 3,1,2,4, respectively. Draw the Gantt chart and calculate the average waiting time & average response time for priority scheduling algorithm (time slicing is denied, the smallest priority number means the highest priority).

在没有时间片的情况下，如果P1、P2、P3、P4的优先级分别是3、1、2、4，请画出优先级调度算法的甘特图、算出其平均等待时间和平均响应时间（优先级号越小的表示优先级越高）。

1. If the time quantum is 2, Draw the Gantt chart and calculate the average waiting time & average response time for Round-Robin algorithm (non-preemptive).

如果时间片是2，在非抢占情况下，请画出轮转算法的甘特图、算出其平均等待时间和平均响应时间。

(1) 甘特图：

|  |  |  |  |
| --- | --- | --- | --- |
| P1 | P2 | P3 | P4 |

0 5 11 15 22

平均等待时间

平均响应时间

(2) 甘特图：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P2 | P4 |

0 5 6 10 15 22

平均等待时间

平均响应时间

(3) 抢占情况下：

甘特图：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P1 | P4 |

0 3 9 13 15 22

平均等待时间

平均响应时间

(4) 甘特图：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P1 | P2 | P1 | P3 | P2 | P4 | P3 | P2 | P4 |

0 2 4 6 7 9 11 13 15 17 22

平均等待时间

平均响应时间

3、

Three process P1, P2 and P3 exclusively use a buffer containing N (N>0) units. P1 uses produce () to generate a positive integer, and uses put () to send it to an empty cell of the buffer at a time; P2 uses getodd () to extract an odd number from the buffer at a time, and uses countodd() to count the odd number; P3 uses geteven () to extract an even number from the buffer at a time, and uses countieven () to count the number of even integers. Please use the semaphore mechanism to achieve synchronization and mutually exclusive activaties of these three processes, and explain the meaning of the defined semaphore. Requires description in pseudocode.

三个进程P1、P2和P3分别使用包含N（N>0）个单元的缓冲区。P1使用produce（）生成正整数，并使用put（）一次将其发送到缓冲区的空单元格；P2使用getodd（）一次从缓冲区提取奇数，并使用count odd（）对奇数进行计数；P3使用geteven（）一次从缓冲区提取偶数，并使用countieven（）对该数进行计数偶数整数。请使用信号量机制来实现这三个进程的同步和互斥激活，并解释定义的信号量的含义，并需要在伪代码中进行必要的解释。

缓冲区是互斥资源，因此设互斥信号量为mutex。同步问题: P1、P2因奇数的放置与取用而同步，设同步信号量为odd; P1、P3因偶数的放置与取用而同步，设同步信号量为even;对空闲缓冲区设置资源同步信号量empty，初值为N。

伪代码描述如下:

semaphore mutex=1; //缓冲区互斥操作信号量

semaphore odd=0,even=0;//奇数、偶数进程的同步信号量

semaphore empty=N; //空缓冲区单元个数信号量

processP1() {

while(true) {

number=produce();

P(empty);

P(mutex);

put();

v(mutex); //释放缓冲区

if(number%2==0) //如果为奇数，则允许取奇数;如果为偶数，则允许取偶数

V(even);

else

V(odd);

}

}

processP2() {

while(true) {

p(odd); //互斥奇数

p(mutex); //互斥访问缓冲区

getodd();

v(mutex); //释放缓冲区

v(empty); //释放一个空缓冲区

countodd();

}

}

processP3() {

while(true) {

p(even);

p(mutex);

geteven();

v(mutex);

v(empty);

counteven();

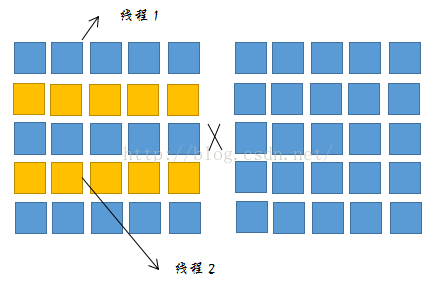
}

}

4、请写出多线程矩阵乘法的主要步骤和核心代码，我们推荐使用pthread API来表示多线程操作，如果使用其他语言/平台的多线程库，请进行相应的解释。Pthread参见《操作系统概念》中文（第9版） 4.4.1节。

提示：答案不唯一，以下给出两种示例的方法

**方法1：对于矩阵乘法A\*B，可以对A做按行分块处理或对B做按列分块处理。**



将A按奇数行和偶数行进行分块操作，奇数行的矩阵乘法通过线程1执行，偶数行的矩阵乘法通过线程2执行。

**方法2：产生n\*n个线程，每组元素相乘a[i][j]\*b[x][y]进入一个线程运算最后返回结果。**

使用方法2的思路。

给定两个矩阵*A*和*B*, 其中*A*是具有*M*行、*K*列的矩阵, *B*为*K*行、*N*列的矩阵, *A*和*B*的矩阵积为矩阵*C*，*C*为*M*行、*N*列。矩阵*C*中第i行、第j列的元素 就是矩阵*A*第i行每个元素和矩阵*B*第j列每个元素乘积的和, 即

每个 的计算用一个独立的工作线程, 因此它将会涉及生成 个工作线程。主线程(或称为父线程)将初始化矩阵 和 , 并分配足够的内存给矩阵 , 它将容纳矩阵 和 的积。这些矩阵将声明为全局数据, 以使每个工作线程都能访问矩阵 和 。

#include <stdio.h>

#include <pthread.h>

#define DIMENSION 10

pthread\_t hThread[DIMENSION][DIMENSION];

int J, K, M, N;

int mat1[DIMENSION][DIMENSION];

int mat2[DIMENSION][DIMENSION];

int result[DIMENSION][DIMENSION];

struct Data{

int row;

int col;

};

void \*ThreadProc(void \*data){

struct Data \*tmp = (struct Data \*)data;

int rs = 0;

int loop;

for(loop = 0; loop < K; loop++){

rs += mat1[tmp->row][loop]\*mat2[loop][tmp->col];

}

result[tmp->row][tmp->col] = rs;

}

int main(){

int i, j;

struct Data data[DIMENSION][DIMENSION];

int ret;

printf("输入矩阵行数和列数：");

scanf("%d%d", &J, &K);

printf("输入矩阵：");

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

for(j = 0; j < K; j++){

scanf("%d", &mat1[i][j]);

}

}

printf("输入矩阵行数和列数：");

scanf("%d%d", &M, &N);

printf("输入矩阵：");

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

for(j = 0; j < N; j++){

scanf("%d", &mat2[i][j]);

}

}

if( K != M || J > 10 || K > 10 || M > 10 || N > 10){

printf("Your input produce an error\n");

return -1;

}

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

for(j = 0; j < N; j++){

data[i][j].row = i;

data[i][j].col = j;

ret = pthread\_create(&hThread[i][j], NULL, ThreadProc, (void\*)&data[i][j]);

if(ret != 0){

printf("Create thread error!\n");

return -1;

}

}

}

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

for(j = 0; j < N; j++){

pthread\_join(hThread[i][j], NULL);

}

}

printf("\n运算结果:\n");

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

for(j = 0; j < N; j++){

printf("%-5d ", result[i][j]);

}

printf("\n");

}

return 0;

}