南京信息工程大学 实验（实习）报告

实验（实习）名称 进程调度算法的模拟 实验（实习）日期 得分 指导教师

系 计算机学院专业 物联网工程年级 2021 班次 2 姓名 学号

1. **实验目的**

通过动态优先权算法的模拟加深对进程概念和进程调度过程的理解。

1. **实验内容与步骤**

(1)实现对N个进程采用某种进程调度算法（如动态优先权调度）的调度；

(2)每个用来标识进程的进程控制块PCB可用结构来描述，包括以下字段：

a) 进程标识数ID；

b) 进程优先数 PRIORITY，并规定优先数越大的进程，其优先权越高；

c) 进程已占用CPU时间 CPUTIME

d) 进程还需占用的CPU时间 ALLTIME，当进程运行完毕时， ALLTIME变为0

e) 进程的阻塞时间 STARTBLOCK，表示当进程再运行 STARTBLOCK个时间片后，进程将进入阻塞状态；

f) 进程被阻塞的时间 BLOCKTIME，表示已阻塞的进程再等待 BLOCKTIME个时间片后，将转换成就绪状态。

g) 进程状态 STATE；

h) 队列指针NEXT，用来将PCB排成队列。

(3)优先数改变的原则：

进程在就绪队列中呆一个时间片，优先数增加1；

进程每运行一个时间片，优先数减3。

(4) 假设在调度前，系统中有5个进程，合理设计它们的初始状态。

一个参考示例如下：

ID 0 1 2 3 4

PRIORITY 9 38 30 29 0

CPUTIME 0 0 0 0 0

ALLTIME 3 3 6 3 4

STARTBLOCK 2 -1 -1 -1 -1

BLOCKTIME 3 0 0 0 0

STATE READY READY READY READY READY

(5) 为了清楚地观察每个进程的调度过程，程序应将每个时间片内的进程的情况显示出来，包括正在运行的进程，处于就绪队列中的进程和处于阻塞队列中的进程。

参照的具体格式如下：

RUNNING PROG： i

READY\_QUEUE:->id1->id2

BLOCK\_QUEUE:->id3->id4

==================================

ID 0 1 2 3 4

PRIORITY P0 P1 P2 P3 P4

CPUTIME C0 C1 C2 C3 C4

ALLTIME A0 A1 A2 A3 A4

STARTBLOCK T0 T1 T2 T3 T4

BLOCKTIME B0 B1 B2 B3 B4

STATE S0 S1 S2 S3 S4

1. **实验步骤**

*// 动态优先权调度算法模拟*

#include <stdio.h>

#include <stdlib.h>

*// 定义进程控制块PCB*

typedef struct pcb {

    int id; *// 进程标识数*

    int priority; *// 进程优先数*

    int cputime; *// 进程已占用CPU时间*

    int alltime; *// 进程还需占用的CPU时间*

    int startblock; *// 进程的阻塞时间*

    int blocktime; *// 进程被阻塞的时间*

    char state; *// 进程状态*

    struct pcb \*next; *// 队列指针*

} PCB;

*// 定义就绪队列和阻塞队列*

PCB \*ready = NULL;

PCB \*block = NULL;

*// 创建一个新的PCB*

PCB \*newPCB(int id, int priority, int alltime, int startblock, int blocktime) {

    PCB \*p = (PCB \*)malloc(sizeof(PCB));

    p->id = id;

    p->priority = priority;

    p->cputime = 0;

    p->alltime = alltime;

    p->startblock = startblock;

    p->blocktime = blocktime;

    p->state = 'R'; *// 初始状态为就绪*

    p->next = NULL;

    return p;

}

*// 将一个PCB插入就绪队列，按优先数降序排列*

void insertReady(PCB \*p) {

    PCB \*prev = NULL;

    PCB \*curr = ready;

    while (curr != NULL && curr->priority >= p->priority) {

        prev = curr;

        curr = curr->next;

    }

    if (prev == NULL) { *// 插入队首*

        p->next = ready;

        ready = p;

    } else { *// 插入中间或队尾*

        p->next = curr;

        prev->next = p;

    }

}

*// 将一个PCB插入阻塞队列，按阻塞时间升序排列*

void insertBlock(PCB \*p) {

    PCB \*prev = NULL;

    PCB \*curr = block;

    while (curr != NULL && curr->blocktime <= p->blocktime) {

        prev = curr;

        curr = curr->next;

    }

    if (prev == NULL) { *// 插入队首*

        p->next = block;

        block = p;

    } else { *// 插入中间或队尾*

        p->next = curr;

        prev->next = p;

    }

}

*// 从就绪队列中删除一个PCB*

PCB \*deleteReady() {

    PCB \*p = ready;

    if (p != NULL) {

        ready = p->next;

        p->next = NULL;

    }

    return p;

}

*// 从阻塞队列中删除一个PCB*

PCB \*deleteBlock() {

    PCB \*p = block;

    if (p != NULL) {

        block = p->next;

        p->next = NULL;

    }

    return p;

}

*// 显示一个PCB的信息*

void displayPCB(PCB \*p) {

    printf("%d\t%d\t%d\t%d\t%d\t%d\t%c\n", p->id, p->priority, p->cputime, p->alltime, p->startblock, p->blocktime, p->state);

}

*// 显示就绪队列和阻塞队列的信息*

void displayQueue() {

    PCB \*p;

    printf("READY\_QUEUE:");

    p = ready;

    while (p != NULL) {

        printf("->%d", p->id);

        p = p->next;

    }

    printf("\n");

    printf("BLOCK\_QUEUE:");

    p = block;

    while (p != NULL) {

        printf("->%d", p->id);

        p = p->next;

    }

    printf("\n");

}

*// 显示所有进程的信息*

void displayAll() {

    printf("ID\tPRIORITY\tCPUTIME\tALLTIME\tSTARTBLOCK\tBLOCKTIME\tSTATE\n");

    PCB \*p;

    p = ready;

    while (p != NULL) {

        displayPCB(p);

        p = p->next;

    }

    p = block;

    while (p != NULL) {

        displayPCB(p);

        p = p->next;

    }

}

*// 模拟一个时间片的运行*

void run() {

    PCB \*p = deleteReady(); *// 从就绪队列中取出一个进程*

    if (p != NULL) {

        printf("RUNNING PROG: %d\n", p->id); *// 显示正在运行的进程*

        displayQueue(); *// 显示队列情况*

        printf("==================================\n");

        p->cputime++; *// 进程已占用CPU时间加1*

        p->alltime--; *// 进程还需占用的CPU时间减1*

        if (p->alltime == 0) { *// 进程运行结束*

            p->state = 'F'; *// 进程状态变为完成*

            printf("Process %d finished.\n", p->id);

            free(p); *// 释放进程空间*

        } else if (p->startblock == p->cputime) { *// 进程进入阻塞状态*

            p->state = 'B'; *// 进程状态变为阻塞*

            printf("Process %d blocked.\n", p->id);

            insertBlock(p); *// 将进程插入阻塞队列*

        } else { *// 进程继续就绪*

            p->state = 'R'; *// 进程状态变为就绪*

            p->priority -= 3; *// 进程优先数减3*

            insertReady(p); *// 将进程插入就绪队列*

        }

    }

}

*// 模拟阻塞队列中的进程等待*

void wait() {

    PCB \*p = block;

    while (p != NULL) {

        p->blocktime--; *// 进程被阻塞的时间减1*

        if (p->blocktime == 0) { *// 进程解除阻塞*

            p->state = 'R'; *// 进程状态变为就绪*

            p->priority += 1; *// 进程优先数加1*

            printf("Process %d unblocked.\n", p->id);

            PCB \*q = p->next; *// 保存下一个进程的指针*

            deleteBlock(); *// 从阻塞队列中删除该进程*

            insertReady(p); *// 将该进程插入就绪队列*

            p = q; *// 继续遍历阻塞队列*

        } else {

            p = p->next; *// 继续遍历阻塞队列*

        }

    }

}

int main() {

*// 创建五个进程，按照题目给出的初始状态*

    PCB \*p0 = newPCB(0, 9, 3, 2, 3);

    PCB \*p1 = newPCB(1, 38, 3, -1, 0);

    PCB \*p2 = newPCB(2, 30, 6, -1, 0);

    PCB \*p3 = newPCB(3, 29, 3, -1, 0);

    PCB \*p4 = newPCB(4, 0, 4, -1, 0);

*// 将五个进程插入就绪队列*

    insertReady(p0);

    insertReady(p1);

    insertReady(p2);

    insertReady(p3);

    insertReady(p4);

*// 显示初始状态*

    printf("Initial state:\n");

    displayQueue();

    displayAll();

    printf("==================================\n");

*// 循环运行，直到就绪队列和阻塞队列都为空*

    while (ready != NULL || block != NULL) {

        run(); *// 模拟一个时间片的运行*

        wait(); *// 模拟阻塞队列中的进程等待*

        displayAll(); *// 显示所有进程的信息*

        printf("==================================\n");

    }

    printf("All processes completed.\n");

    return 0;

}

