四川大学计算机学院、软件学院

实验报告

学号：2022141460155 姓名：林诺晗 专业：计算金融 班级：计金班 第10 周

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 课程名称 | 操作系统实验 | | 实验课时 | 2 |
| 实验项目 | 进程的调度 | | 实验时间 | 2024.04.28 |
| 实验目的 | （1）加深对进程概念的理解，明确进程和程序的区别  （2）深入理解系统如何组织进程  （3）理解常用进程调度算法的具体实现。 | | | |
| 实验环境 | VMware Workstation 17 Pro  Ubuntu 22.04.3 LTS | | | |
| 实验内容（算法、程序、步骤和方法） | 编写C程序模拟实现单处理机系统中的进程调度算法，实现对多个进程的调度模拟，要求采用常见进程调度算法（如先来先服务、优先级调度等算法）进行模拟调度。  **实验相关原理：Linux进程调度**  Linux 中有一个总的调度结构，称之为调度器类（scheduler class）  允许不同的可动态添加的调度算法并存，总调度器根据调度器类的优先顺序，依次挑选调度器类中的进程进行调度。  确定调度器类后，再使用该调度器类的调度算法（调度策略）进行内部调度。  调度器类的优先级顺序为：  Stop\_Task > Real\_Time > Fair > Idle\_Task  其中，Fair和Real\_time最常用，分别采用CFS（完全公平调度算法）调度算法的默认调度类和实时调度类。  **实验相关要求：**  数据结构设计：  PCB：结构体  就绪队列：链表，每个节点为进程PCB  进程状态  调度算法设计：  具体调度算法：FCFS、PR  涉及多种操作：排序、链表操作  程序的输出：  调度进程的顺序、每个进程的起始时间、终止时间等  CPU每次调度的过程  **源代码：**  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  typedef struct Process {  char\* name;  int priority;  int runtime;  int time\_in\_queue;  struct Process\* next;  } Process;  Process\* createProcess(const char\* name, int priority, int runtime) {  Process\* newProcess = (Process\*)malloc(sizeof(Process));  if (newProcess == NULL) {  perror("Memory allocation failed");  exit(EXIT\_FAILURE);  }  newProcess->name = strdup(name);  newProcess->priority = priority;  newProcess->runtime = runtime;  newProcess->time\_in\_queue = 0;  newProcess->next = NULL;  return newProcess;  }  void enqueue(Process\*\* head, Process\* process) {  process->next = \*head;  \*head = process;  }  Process\* dequeue(Process\*\* head) {  Process\* process = \*head;  \*head = (\*head)->next;  return process;  }  int compareProcesses(const void\* a, const void\* b) {  Process\* processA = \*(Process\*\*)a;  Process\* processB = \*(Process\*\*)b;  return processB->priority - processA->priority;  }  void simulateScheduling(Process\*\* readyQueue) {  Process\* current = \*readyQueue;  int time = 0;  while (current != NULL) {  Process\* runningProcess = dequeue(readyQueue);  runningProcess->time\_in\_queue = time++;  printf("The execute number:%d\n", time);  printf("\*\*\*\*\n");  printf("当前正在运行的进程是：%s\n", runningProcess->name);  printf("\*\*\*\*当前就绪队列状态为：\n");  Process\* process = \*readyQueue;  while (process != NULL) {  printf("qname\tstate\tnice\tndtime\truntime\n");  printf("%s\tW\t%d\t%d\t%d\n", process->name, process->priority, process->time\_in\_queue, process->runtime);  process = process->next;  }  printf("按任一键继续：");  getchar(); // 等待用户输入以继续  if (--runningProcess->runtime > 0) {  enqueue(readyQueue, runningProcess); // 如果进程尚未完成，将其重新加入就绪队列  } else {  free(runningProcess->name);  free(runningProcess);  }  }  }  int main() {  Process\* readyQueue = NULL;  int num\_processes;  printf("请输入被调度的进程数目：");  scanf("%d", &num\_processes);  for (int i = 0; i < num\_processes; i++) {  char name[50];  int priority, runtime;  printf("进程号No.%d：\n", i);  printf("输入进程名：");  scanf("%s", name);  printf("输入进程优先数：");  scanf("%d", &priority);  printf("输入进程运行时间：");  scanf("%d", &runtime);  enqueue(&readyQueue, createProcess(name, priority, runtime));  }  simulateScheduling(&readyQueue);  while (readyQueue != NULL) {  Process\* temp = readyQueue;  readyQueue = readyQueue->next;  free(temp->name);  free(temp);  }  return 0;  }  rocess 结构体用于表示一个进程控制块（PCB）。  createProcess 函数用于创建一个新的进程。  compareProcesses 函数用于比较两个进程的优先级。  enqueue 函数用于将新进程添加到就绪队列的头部。  dequeue 函数用于从就绪队列的头部移除并返回一个进程。  simulateScheduling 函数是调度模拟的核心，它按照以下步骤执行：  --》将就绪队列中的所有进程复制到一个数组中。  --》使用 qsort 和 compareProcesses 对数组中的进程按照优先级进行排序。  --》遍历排序后的进程数组，模拟每个进程的执行。  1  2  ****编译程序****：使用编译器（如gcc）将.c文件编译成可执行文件。  ****运行程序****：编译成功后，会生成一个可执行文件，程序运行如下：  3 | | | |
| 结论  （结果） | 实验成功，按照相关要求完成程序编写及实现，并熟练掌握了相关命令的使用。  模拟实现了单处理机系统中的进程调度算法，实现了对多个进程的调度模拟。 | | | |
| 小结 | （1）加深了对进程概念的理解，明确进程和程序的区别  （2）深入理解了系统如何组织进程  （3）理解了常用进程调度算法的具体实现。 | | | |
| 指导老师  评议 |  | | | |
| 成绩评定： | 指导教师签名： | | |