实验三实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程名称** | 操作系统 | **指导教师** | 叶晖 | **实验日期** | 2022.04.25 |
| **班级** | 20软件2班 | **姓名** | 李佳骏 | **学号** | B20190103224 |
| **实验成绩** |  | | | | |
| **一、实验名称**  进程调度 | | | | | |
| **二、实验目的**  (1) 了解进程调度的几种算法，并比较各种调度算法的性能。 | | | | | |
| **三、实验环境**  (1) 硬件环境：Intel Pentium III 以上 CPU，128MB 以上内存，2GB 以上硬盘  (2) 软件环境：Ubuntu 13.04 操作系统。  注：Ubuntu 是 Linux 的一个免费发行的流行版本，系英国"科能软件股份有限公司  (Canonical)"开发 | | | | | |
| **四、实验内容**  (1) 熟悉几种调度算法；  (2) 根据演示的程序结果，分析程序的采用的是哪种调度算法；  (3) 编写 FIFO，SJF, SRTF, 动态优先级调度算法的程序，并计算其周转时间，平均作业周转时间，平均带权作业周转时间。 | | | | | |
| **五、实验原理**  (1) 实验原理：操作系统的调度程序担负两项任务：调度和分派。前者实现调度策略，确定  就绪进程/线程竞争使用处理器的次序的裁决原则，即进程/线程何时应放弃 CPU 和选择哪  个进程/线程来执行；后者执实现调度机制，确定如何时分复用 CPU，处理上下文细节，完成  进程/线程同 CPU 的绑定及放弃的实际工作。  (2) 知识点:先来先服务算法(FIFO)、最短作业调度算法(SJF)、最短剩余时间优先算法  (SRTF)、响应比最高者优先算法(HRRF)、优先级调度算法(静态优先级和动态优先级调度算法)、轮转调度算法(RR)、多级反馈队列调度算法(MLFQ)、彩票调度算法。 | | | | | |
| **六、实验步骤**  【**任务** 】  (1) 程序设计：设计一个调度算法，能够显示一个进程调度的方式，并且将其调度处理结果保存至文件中。  (2)上机操作  用 gedit 输入源代码。  编译、运行，观察屏幕，记录结果。  思考:请先运行此程序，查看运行结果，分析此调度属于哪种调度算法。描述此种调度算法的调度过程  **代码如下图：**  #include <stdio.h>  #include <unistd.h>  #include<stdlib.h>  #include <string.h>  typedef struct pcb  {  int pid;  int prior;  char status[10];  int time;  struct pcb \*next;  } PCB;  FILE \*fp = NULL;  int main()  {  int iNumProc = 0;  int i = 0;  PCB \*stPCBHead = NULL;  PCB \*stPCB = NULL;  PCB \*stPCBFront = NULL;  PCB \*stPCBNode = NULL;  char filename[128];  memset(filename, 0x00, sizeof(filename));  sprintf(filename, "/home/ljjtpcn/Desktop/schedule.txt");  if ((fp = fopen(filename, "w")) == NULL)  {  printf("Create log file failed!\n");  return -1;  }  /\* Input the number of process \*/  printf("Please input the number of process\n");  scanf("%d", &iNumProc);  /\* Process number must great than 0 \*/  while (iNumProc <= 0)  {  printf("Ther number of process cann't be 0! \n");  printf("Please input the number of process again\n");  scanf("%d", &iNumProc);  }  for (i = 0; i < iNumProc; i++)  {  /\* Create process's node \*/  stPCBNode = (PCB \*)malloc(sizeof(PCB));  if (stPCBNode == NULL)  {  printf("malloc memory failed!\n");  return -1;  }  memset(stPCBNode, 0x00, sizeof(PCB));  /\* Init process's information \*/  stPCBNode->pid = i; /\* Process ID \*/  printf("Please input ther prior of process[%d]\n", i);  scanf("%d", &stPCBNode->prior); /\* Process's prior \*/  while (stPCBNode->prior < 0)  {  printf("Process's prior must great than 0\n");  printf("Please input ther run time of process[%d]\n", i);  scanf("%d", &stPCBNode->prior);  }  printf("Please input ther run time of process[%d][%d]\n", i, \_\_LINE\_\_);  scanf("%d", &stPCBNode->time);  while (stPCBNode->time < 0)  {  printf("Process's run time must great than 0\n");  printf("Please input ther run time of process[%d]\n", i);  scanf("%d", &stPCBNode->time);  }  strcpy(stPCBNode->status, "ready"); /\* Init the process's status ready  \*/  stPCBNode->next = NULL;  /\* Link the new process's node \*/  if (stPCBHead == NULL)  {  stPCBHead = stPCBNode;  continue;  }  stPCB = stPCBHead;  stPCBFront = stPCBHead;  while (stPCB != NULL && stPCB->prior >= stPCBNode->prior)  {  stPCBFront = stPCB;  stPCB = stPCB->next;  }  if (stPCB == stPCBHead)  {  stPCBNode->next = stPCBHead;  stPCBHead = stPCBNode;  }  else if (stPCB == NULL)  {  stPCBFront->next = stPCBNode; /\* Add to the tail of the link  \*/  }  else /\* insert into the link \*/  {  stPCBNode->next = stPCB;  stPCBFront->next = stPCBNode;  }  printf("Create the process [%d] success\n", i);  }  fprintf(fp, "Create %d processes success\n", iNumProc);  printf("In the begin of schedule, these process's queque\n");  fprintf(fp, "Before Schedule\n");  printProc(stPCBHead);  sleep(1);  /\* Now Schedual \*/  printf("Begin schedule\n");  fprintf(fp, "Begin schedule\n");  stPCB = stPCBHead;  while (iNumProc > 0)  {  /\* schedule from first process \*/  if (strcmp(stPCBHead->status, "ready") == 0)  strcpy(stPCBHead->status, "running");  printProc(stPCBHead);  stPCBHead->time--;  stPCBHead->prior--;  /\* Update the the level of proccess which in wait status \*/  for (stPCB = stPCBHead; stPCB != NULL; stPCB = stPCB->next)  {  if (strcmp(stPCB->status, "ready") == 0)  {  stPCB->prior++;  }  }  /\* sort the schedule again \*/  sort(&stPCBHead, &iNumProc); /\* Sort the link and delete which the  process's time is 0 \*/  }  return 0;  }  int sort(PCB \*\*pstPCBHead, int \*iNumProc)  {  PCB \*pstPCB = NULL;  PCB \*pstPCBFront = NULL;  int flag = 0;  pstPCB = (\*pstPCBHead)->next;  pstPCBFront = (\*pstPCBHead);  if ((\*pstPCBHead)->time == 0)  {  printf("Proccess [%d] run end!\n", pstPCBFront->pid);  (\*pstPCBHead) = (\*pstPCBHead)->next;  free(pstPCBFront);  pstPCBFront = NULL;  (\*iNumProc)--;  return 0;  }  if ((\*iNumProc) <= 1)  {  return 0;  }  while (pstPCB != NULL && (\*pstPCBHead)->prior <= pstPCB->prior)  {  pstPCBFront = pstPCB;  pstPCB = pstPCB->next;  flag = 1;  }  if (pstPCB == NULL && flag == 1)  {  strcpy((\*pstPCBHead)->status, "ready");  pstPCBFront->next = (\*pstPCBHead);  \*pstPCBHead = (\*pstPCBHead)->next;  pstPCBFront->next->next = NULL;  }  else if (flag == 1)  {  strcpy((\*pstPCBHead)->status, "ready");  pstPCBFront->next = (\*pstPCBHead);  (\*pstPCBHead) = (\*pstPCBHead)->next;  pstPCBFront->next->next = pstPCB;  }  return 0;  }  int printProc(PCB \*pstPCBHead)  {  PCB \*pstPCB=pstPCBHead;  printf("-----------------------------------------------------------\n");  printf("| Process's ID | Process's prior| Process's Stauts |The time left|Current Process Addr|Next Process Addr|\n");  printf("-----------------------------------------------------------\n");  fprintf(fp, "---------------------------------------------------------\n");  fprintf(fp, "| Process's ID | Process's prior| Process's Stauts |The time left|Current Process Addr|Next Process Addr|\n");  fprintf(fp, "-----------------------------------------------------------\n");  while ( pstPCB != NULL )  {  sleep(1);  printf("|%16d|%16d|%18s|%13d|%20d|%17d|\n", pstPCB->pid,  pstPCB->prior, pstPCB->status, pstPCB->time, pstPCB, pstPCB->next);  printf("------------------------------------------------------\n");  fprintf(fp, "|%16d|%16d|%18s|%13d|%20d|%17d|\n", pstPCB->pid,  pstPCB->prior, pstPCB->status, pstPCB->time, pstPCB, pstPCB->next);  fprintf(fp, "--------------------------------------------\n");  pstPCB=pstPCB->next;  }  printf("\n\n");  fprintf(fp, "\n\n");  return 0;  } | | | | | |
| **七、实验结果**  本次测试输入4个进程，分别设置2,5,3,4优先度和2,3,5,9运行时间    **运行步骤如下所示：**  进程1最先运行结束    其次是进程0    再后是进程2    最后是进程3 | | | | | |
| **八、实验总结**  从4个进程的运行次序可以看出，每次运行时，正在运行的进程优先度减一，等待的进程优先级加一。  这种调度算法使用了优先级这一评判次序的指标，并且优先级随着时间的变化而产生变化，所以它是动态优先级算法。  通过本次试验理解了各种调度算法以及它们的优劣  程序解析：  (1)pcb结构体:即一个进程，包含进程的所有信息和指向下一个进程的指针  (2)printproc函数:用于整理输出信息，从左到右分别是程序号，优先级，状态，剩余需要时间，地址，下一个进程的地址  (3)sort()函数:用于按照动态优先级调度算法对pcb结构体排序  (4)main函数流程:创建文件->输入进程信息->按优先级连接各进程->输出整理好的信息->while循环排序和输出直到全部进程运行结束 | | | | | |
| 1. **教师评阅意见** | | | | | |