

武汉大学计算机学院

本科生实验报告

操作系统设计与实现

专 业 名 称 ： 软件工程卓越工程师

课 程 名 称 ： 操作系统设计

指 导 教 师 ： 曾平

学 生 学 号 ： 2019302110166

学 生 姓 名 ： 陈子懿

二〇二一年六月

郑 重 声 明

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名：_____ 日期：_____

摘 要

操作系统设计与实现实验的实验目的是巩固《操作系统》课中所学到的关于 CPU 调度、内存分配、文件管理等等知识，并且运用这些理论知识来模拟操作系统内核所实现的功能。

关键词：处理机；主存储器；磁盘；同步

目录

武汉大学计算机学院.....	1
本科生实验报告.....	1
郑 重 声 明.....	2
1 处理机调度.....	5
1.1 实验内容及上机实验所用平台.....	5
1.2 数据结构及代码段分析.....	5
1.3 调试过程.....	7
1.4 实验总结.....	15
2 主存空间的分配和回收.....	17
2.1 实验内容及上机实验所用平台.....	17
2.2 数据结构及代码段分析.....	17
2.3 调试过程.....	21
2.4 实验总结.....	22
3 磁盘存储空间的分配和回收.....	24
3.1 实验内容及上机实验所用平台.....	24
3.2 数据结构及代码段分析.....	24
3.3 调试过程.....	27
3.4 实验总结.....	31
4 进程创建.....	32
4.1 实验内容及上机实验所用平台.....	32
4.2 数据结构及代码段分析.....	32
4.3 调试过程.....	33
4.4 实验总结.....	33
5 进程同步.....	35
5.1 实验内容及上机实验所用平台.....	35
5.2 数据结构及代码段分析.....	35
5.3 调试过程.....	42
5.4 实验总结.....	47
参考文献.....	48

1 处理机调度

1.1 实验内容及上机实验所用平台

选择一个调度算法，实现处理器调度。

1. 1. 1 实验内容

通过实习模拟在单处理器环境下的处理器调度，加深了解处理器调度的工作。

本实验采用高优先级优先调度算法。

1. 1. 2 上机平台

系统：Windows

IDE：vscode

1.2 数据结构及代码段分析

进程的状态分为就绪和完成：

```
1. enum PROCESS_STATUS{
2.     READY, ZOMBIE
3. };
```

每个进程包含一个 PCB 进程控制块，其中包含了

- 进程名——如 P1~P5。
- 指针——按优先数的大小把 5 个进程连成队列，用指针指出下一个进程 PCB 的首地址。
- 要求运行时间——假设进程需要运行的单位时间数。
- 优先数——赋予进程的优先数，调度时总是选取优先数大的进程先执行。
- 状态——假设两种状态：就绪和结束，用 R 表示就绪，用 E 表示结束。初始状态都为就绪状态。

```
1. struct PCB{
2.     bool operator<(const PCB& a) const{
3.         return priority < a.priority; //大顶堆
4.     }
```

```

5.     string processName;
6.     int priority;
7.     // PCB* nxtPCB;
8.     int demandTime;
9.     PROCESS_STATUS status;
10. }pool[5];

```

就绪进程队列是一个大根堆，在 PCB 结构体中已经实现了对于<的重载：

```

1. priority_queue<PCB> readyQueue;

```

打印就绪队列：

```

4. void printReadyQueue(){
5.     priority_queue<PCB> readyQueueClone = readyQueue;
6.     printf("\nreadyQueue:\n");
7.     printf("ProcessName\tPriority\tdemandTime\t\n");
8.     printf("-----\n");
9.     while(!readyQueueClone.empty()){
10.         PCB x = readyQueueClone.top();
11.         readyQueueClone.pop();
12.         printf("%s\t\t%d\t\t%d\t\n", x.processName.c_str(), x.priority, x.demandTime);
13.     }
14.     printf("\n");
15. }

```

开始执行进程：

```

16. void start(){
17.     while(!readyQueue.empty()){
18.         printReadyQueue();
19.         PCB x = readyQueue.top();
20.         readyQueue.pop();
21.         cout << "currently running: " << x.processName << endl;
22.         cout << "remaining demand time: " << --x.demandTime << endl;
23.         cout << "priority after running: " << --x.priority << endl;
24.         if(x.demandTime <= 0) x.status = ZOMBIE;
25.         else readyQueue.push(x);
26.         system("pause");
27.     }
28. }

```

Main 函数，初始化进程参数：

```

29. int main(){
30.     //10 5freopen("h.in", "r", stdin);
31.     cout << "please input the priority and demand time of each process." <<
        endl;
32.     for(int i = 0; i <= 4; i++){
33.         int pri, dt;
34.         cin >> pri >> dt;
35.         pool[i].priority = pri;
36.         pool[i].demandTime = dt;
37.         std::ostringstream oss;
38.         oss << "P " << i;
39.         std::string str = oss.str();
40.         pool[i].processName = str;
41.
42.         pool[i].status = READY;
43.         cout<<pool[i].processName<<endl;
44.         readyQueue.push(pool[i]);
45.
46.     }
47.     start();
48.     return 0;
49. }

```

1.3 调试过程

进入 Windows/Linux 系统，输入 `g++ priority.cpp -o priority`
 然后运行 `./priority`
 运行结果如下：

```

1. PS D:\Study\大学资料\操作系统\实验\1\1> .\priority.exe
2. please input the priority and demand time of each process.
3. 10 5
4. P 0
5. 4 6
6. P 1
7. 7 3
8. P 2
9. 5 8
10. P 3
11. 2 6
12. P 4
13.

```

```

14. readyQueue:
15. ProcessName      Priority      demandTime
16. -----
17. P 0              10          5
18. P 2              7          3
19. P 3              5          8
20. P 1              4          6
21. P 4              2          6
22.
23. currently running: P 0
24. remaining demand time: 4
25. priority after running: 9
26. 请按任意键继续. . .
27.
28. readyQueue:
29. ProcessName      Priority      demandTime
30. -----
31. P 0              9          4
32. P 2              7          3
33. P 3              5          8
34. P 1              4          6
35. P 4              2          6
36.
37. currently running: P 0
38. remaining demand time: 3
39. priority after running: 8
40. 请按任意键继续. . .
41.
42. readyQueue:
43. ProcessName      Priority      demandTime
44. -----
45. P 0              8          3
46. P 2              7          3
47. P 3              5          8
48. P 1              4          6
49. P 4              2          6
50.
51. currently running: P 0
52. remaining demand time: 2
53. priority after running: 7
54. 请按任意键继续. . .
55.
56. readyQueue:
57. ProcessName      Priority      demandTime

```


58. -----

59. P 2	7	3
---------	---	---

60. P 0	7	2
---------	---	---

61. P 3	5	8
---------	---	---

62. P 1	4	6
---------	---	---

63. P 4	2	6
---------	---	---

64.

65. currently running: P 2

66. remaining demand time: 2

67. priority after running: 6

68. 请按任意键继续. . .

69.

70. readyQueue:

71. ProcessName	Priority	demandTime
-----------------	----------	------------

72. -----

73. P 0	7	2
---------	---	---

74. P 2	6	2
---------	---	---

75. P 3	5	8
---------	---	---

76. P 1	4	6
---------	---	---

77. P 4	2	6
---------	---	---

78.

79. currently running: P 0

80. remaining demand time: 1

81. priority after running: 6

82. 请按任意键继续. . .

83.

84. readyQueue:

85. ProcessName	Priority	demandTime
-----------------	----------	------------

86. -----

87. P 2	6	2
---------	---	---

88. P 0	6	1
---------	---	---

89. P 3	5	8
---------	---	---

90. P 1	4	6
---------	---	---

91. P 4	2	6
---------	---	---

92.

93. currently running: P 2

94. remaining demand time: 1

95. priority after running: 5

96. 请按任意键继续. . .

97.

98. readyQueue:

99. ProcessName	Priority	demandTime
-----------------	----------	------------

100. -----

101. P 0	6	1
----------	---	---

```

102. P 3          5          8
103. P 2          5          1
104. P 1          4          6
105. P 4          2          6
106.
107. currently running: P 0
108. remaining demand time: 0
109. priority after running: 5
110. 请按任意键继续. . .
111.
112. readyQueue:
113. ProcessName    Priority    demandTime
114. -----
115. P 3            5          8
116. P 2            5          1
117. P 1            4          6
118. P 4            2          6
119.
120. currently running: P 3
121. remaining demand time: 7
122. priority after running: 4
123. 请按任意键继续. . .
124.
125. readyQueue:
126. ProcessName    Priority    demandTime
127. -----
128. P 2            5          1
129. P 1            4          6
130. P 3            4          7
131. P 4            2          6
132.
133. currently running: P 2
134. remaining demand time: 0
135. priority after running: 4
136. 请按任意键继续. . .
137.
138. readyQueue:
139. ProcessName    Priority    demandTime
140. -----
141. P 1            4          6
142. P 3            4          7
143. P 4            2          6
144.
145. currently running: P 1

```

146. remaining demand time: 5

147. priority after running: 3

148. 请按任意键继续. . .

149.

150. readyQueue:

151. ProcessName	Priority	demandTime
------------------	----------	------------

152. -----		
------------	--	--

153. P 3	4	7
----------	---	---

154. P 1	3	5
----------	---	---

155. P 4	2	6
----------	---	---

156.

157. currently running: P 3

158. remaining demand time: 6

159. priority after running: 3

160. 请按任意键继续. . .

161.

162. readyQueue:

163. ProcessName	Priority	demandTime
------------------	----------	------------

164. -----		
------------	--	--

165. P 1	3	5
----------	---	---

166. P 3	3	6
----------	---	---

167. P 4	2	6
----------	---	---

168.

169. currently running: P 1

170. remaining demand time: 4

171. priority after running: 2

172. 请按任意键继续. . .

173.

174. readyQueue:

175. ProcessName	Priority	demandTime
------------------	----------	------------

176. -----		
------------	--	--

177. P 3	3	6
----------	---	---

178. P 4	2	6
----------	---	---

179. P 1	2	4
----------	---	---

180.

181. currently running: P 3

182. remaining demand time: 5

183. priority after running: 2

184. 请按任意键继续. . .

185.

186. readyQueue:

187. ProcessName	Priority	demandTime
------------------	----------	------------

188. -----		
------------	--	--

189. P 4	2	6
----------	---	---

```

190. P 1          2          4
191. P 3          2          5
192.
193. currently running: P 4
194. remaining demand time: 5
195. priority after running: 1
196. 请按任意键继续. . .
197.
198. readyQueue:
199. ProcessName    Priority    demandTime
200. -----
201. P 1          2          4
202. P 3          2          5
203. P 4          1          5
204.
205. currently running: P 1
206. remaining demand time: 3
207. priority after running: 1
208. 请按任意键继续. . .
209.
210. readyQueue:
211. ProcessName    Priority    demandTime
212. -----
213. P 3          2          5
214. P 4          1          5
215. P 1          1          3
216.
217. currently running: P 3
218. remaining demand time: 4
219. priority after running: 1
220. 请按任意键继续. . .
221.
222. readyQueue:
223. ProcessName    Priority    demandTime
224. -----
225. P 4          1          5
226. P 1          1          3
227. P 3          1          4
228.
229. currently running: P 4
230. remaining demand time: 4
231. priority after running: 0
232. 请按任意键继续. . .
233.

```

```

234. readyQueue:
235. ProcessName      Priority      demandTime
236. -----
237. P 1              1          3
238. P 3              1          4
239. P 4              0          4
240.
241. currently running: P 1
242. remaining demand time: 2
243. priority after running: 0
244. 请按任意键继续. . .
245.
246. readyQueue:
247. ProcessName      Priority      demandTime
248. -----
249. P 3              1          4
250. P 4              0          4
251. P 1              0          2
252.
253. currently running: P 3
254. remaining demand time: 3
255. priority after running: 0
256. 请按任意键继续. . .
257.
258. readyQueue:
259. ProcessName      Priority      demandTime
260. -----
261. P 4              0          4
262. P 1              0          2
263. P 3              0          3
264.
265. currently running: P 4
266. remaining demand time: 3
267. priority after running: -1
268. 请按任意键继续. . .
269.
270. readyQueue:
271. ProcessName      Priority      demandTime
272. -----
273. P 1              0          2
274. P 3              0          3
275. P 4              -1         3
276.
277. currently running: P 1

```

278. remaining demand time: 1

279. priority after running: -1

280. 请按任意键继续. . .

281.

282. readyQueue:

283. ProcessName	Priority	demandTime
------------------	----------	------------

284. -----		
------------	--	--

285. P 3	0	3
----------	---	---

286. P 4	-1	3
----------	----	---

287. P 1	-1	1
----------	----	---

288.

289. currently running: P 3

290. remaining demand time: 2

291. priority after running: -1

292. 请按任意键继续. . .

293.

294. readyQueue:

295. ProcessName	Priority	demandTime
------------------	----------	------------

296. -----		
------------	--	--

297. P 4	-1	3
----------	----	---

298. P 1	-1	1
----------	----	---

299. P 3	-1	2
----------	----	---

300.

301. currently running: P 4

302. remaining demand time: 2

303. priority after running: -2

304. 请按任意键继续. . .

305.

306. readyQueue:

307. ProcessName	Priority	demandTime
------------------	----------	------------

308. -----		
------------	--	--

309. P 1	-1	1
----------	----	---

310. P 3	-1	2
----------	----	---

311. P 4	-2	2
----------	----	---

312.

313. currently running: P 1

314. remaining demand time: 0

315. priority after running: -2

316. 请按任意键继续. . .

317.

318. readyQueue:

319. ProcessName	Priority	demandTime
------------------	----------	------------

320. -----		
------------	--	--

321. P 3	-1	2
----------	----	---

```

322. P 4          -2          2
323.
324. currently running: P 3
325. remaining demand time: 1
326. priority after running: -2
327. 请按任意键继续. . .
328.
329. readyQueue:
330. ProcessName    Priority    demandTime
331. -----
332. P 4          -2          2
333. P 3          -2          1
334.
335. currently running: P 4
336. remaining demand time: 1
337. priority after running: -3
338. 请按任意键继续. . .
339.
340. readyQueue:
341. ProcessName    Priority    demandTime
342. -----
343. P 3          -2          1
344. P 4          -3          1
345.
346. currently running: P 3
347. remaining demand time: 0
348. priority after running: -3
349. 请按任意键继续. . .
350.
351. readyQueue:
352. ProcessName    Priority    demandTime
353. -----
354. P 4          -3          1
355.
356. currently running: P 4
357. remaining demand time: 0
358. priority after running: -4
359. 请按任意键继续. . .

```

1.4 实验总结

- 优先级调度（具有相同优先级的按 FCFS 调度）

- 在作业调度中，从后备作业队列中选择若干优先级高的作业调入内存。
- 在进程调度中，将处理机分配给就绪队列中优先级最高的进程。
- 调度方式分类
 - ◆ 非抢占式
 - ◆ 抢占式
- 优先级分类
- 静态：创建进程时确定
 - ◆ 特点：简单易行，系统开销小，但不精确
- 动态：进程运行过程中更改
 - ◆ 问题：饥饿，低优先级的进程可能永远得不到运行
 - ◆ 解决方案：老化，视进程等待时间的延长提高其优先级

2 主存空间的分配和回收

2.1 实验内容及上机实验所用平台

2.1.1 实验内容

主存储器空间的分配和回收。

本实验采用的时可变分区分配方式。

2.1.2 上机平台

系统: Windows

IDE: vscode

2.2 数据结构及代码段分析

空闲块结构体，包括了空闲块的 index，大小，始址：

```
1. struct Item{
2.     int index;
3.     int sizeInByte;
4.     int startAddr;
5.     bool operator < (int a) const{
6.         return startAddr < a;
7.     }
8. };
```

所有的空闲块

```
9. vector<Item> availTable;
```

所有的作业：

```
10. map<int, Item> jobs;
```

内存每一字节的占用情况，采用差分法，申请空间时，在始址的地方+1，始址+大小的地方+1，这样逐位求出前缀和就能获得内存每一位的占用情况：

```
11. const int MAXN = 128;
12. int memory[MAXN], memoryPrefixSum[MAXN];
```

初始化，先放入一块完整的空闲块。

```

13. void init(){
14.     availTable.clear();
15.     availTable.push_back(Item{1, MAXN, 0});
16.     memset(memory, 0, sizeof memory);
17.     memset(memoryPrefixSum, 0, sizeof memoryPrefixSum);
18. }

```

更新空闲块表，主要工作是合并邻接空闲块，并且将空闲块按照开始地址升序排列：

```

19. void updateAvailTable(){
20.     int curI = 1;
21.     bool suc = true;
22.     for(auto i = availTable.begin(); i != availTable.end(); i++){
23.         if(i->sizeInByte == 0) availTable.erase(i);
24.         else{
25.             i->index = curI++;
26.             if(i!=availTable.begin()){
27.                 if(i->startAddr == (i - 1)->startAddr+(i-1)->sizeInByte){
28.                     int str = (i - 1)->startAddr, size = (i-1)->sizeInByte+i->
sizeInByte;
29.                     *i = Item{0, size, str};
30.                     availTable.erase(i - 1);
31.                     suc = false;
32.                     break;
33.                 }
34.             }
35.         }
36.     }
37.     if(!suc) updateAvailTable();
38. }

```

请求空间，内存标记方法已经在上面说明；此外还要记录该作业对应的是哪块：

```

39. bool askForSpace(int index, int byte){
40.     for(auto i = availTable.begin(); i != availTable.end(); i++){
41.         if(i->sizeInByte >= byte){
42.             i->sizeInByte -= byte;
43.
44.             jobs[index] = Item{index, byte, i->startAddr};
45.
46.             memory[i->startAddr]++;
47.             memory[i->startAddr + byte]--;
48.
49.             i->startAddr += byte;
50.             if(!i->sizeInByte) updateAvailTable();

```

```
51.  
52.         return true;  
53.     }  
54. }  
55. return false;  
56. }  
57.
```

打印出现在所有的空闲块信息:

```
58. void displayAvailTable(){  
59.     cout << "-----"<<endl;  
60.     cout << "index\t|\tsize\t|\tstart\t|\t" << endl;  
61.     for(auto i = availTable.begin(); i != availTable.end(); i++){  
62.         cout << i->index << "\t|\t" << i->sizeInByte<<"\t|\t" << i->startAddr<<  
        "\t|\t\n";  
63.     }  
64.     cout << "-----"<<endl;  
65. }
```

展示出当前内存的占用情况:

```
66. void displayMemory(){
67.     memoryPrefixSum[0] = memory[0];
68.     if(memoryPrefixSum[0]) cout << "#" ;
69.     else cout << "=";
70.     for(int i = 1; i < MAXN; i++){
71.         memoryPrefixSum[i] = memory[i] + memoryPrefixSum[i - 1];
72.         if(memoryPrefixSum[i]) cout << "#" ;
73.         else cout << "=";
74.     }
75.     cout << endl;
76. }
```

释放某作业占用的空间:

```
77. void releaseMemory(int index){
78.     int start = jobs[index].startAddr;
79.     int size = jobs[index].sizeInByte;
80.     memory[start]--;
81.     memory[start + size]++;
82.     auto pos = lower_bound(availTable.begin(), availTable.end(), start);
83.     availTable.insert(pos, Item{0, size, start});
84.     updateAvailTable();
85.     // return true;
```

```
86. }
```

Main 函数，接收来自键盘的命令，执行相应操作：

```
87. int n;
88. int main(){
89.     //freopen("h.in", "r", stdin);
90.     init();
91.     displayMemory();
92.     displayAvailTable();
93.     //return 0;
94.     cout << "how many opt?." << endl;
95.     cin >> n;
96.     for(int i = 1; i <= n; i++){
97.         cout <<endl<< "opt"<<i<<":"<<endl;
98.         char opt;
99.         cin >> opt;
100.        switch(opt){
101.            case 'A':{
102.                int size;
103.                cin >> size;
104.                if(askForSpace(i, size)){
105.                    cout << "ok!";
106.                }
107.            }
108.            else{
109.                cout << "damn!";
110.            }
111.            break;
112.            case 'R':{
113.                int index;
114.                cin >> index;
115.                releaseMemory(index);
116.                break;
117.            }
118.        }
119.        displayMemory();
120.        displayAvailTable();
121.    }
122.    return 0;
123. }
```

2.3 调试过程

进入 Windows/Linux 系统，输入 `g++ dynamic.cpp -o dynamic`

运行 `./dynamic`

运行结果如下：

```
1. PS D:\Study\大学资料\操作系统\实验\2\1> .\dynamic.exe
2. =====
   =====
3. -----
4. index |      sze |      start |
5. 1      |     128 |         0   |
6. -----
7. how many opt?.
8. 5
9.
10. opt1:
11. A 50
12. ok!#####
    =====
13. -----
14. index |      sze |      start |
15. 1      |     78   |        50   |
16. -----
17.
18. opt2:
19. A 30
20. ok!#####
    #####
21. -----
22. index |      sze |      start |
23. 1      |     48   |        80   |
24. -----
25.
26. opt3:
27. A 10
28. ok!#####
    #####
29. -----
30. index |      sze |      start |
31. 1      |     38   |        90   |
32. -----
33.
34. opt4:
```

```

35. R 2
36. #####-----
    ===#####
37. -----
38. index |      size |      start |
39. 1      |      30   |      50    |
40. 2      |      38   |      90    |
41. -----
42.
43. opt5:
44. R 1
45. =====
    ===#####
46. -----
47. index |      size |      start |
48. 1      |      80   |      0     |
49. 2      |      38   |      90    |
50. -----

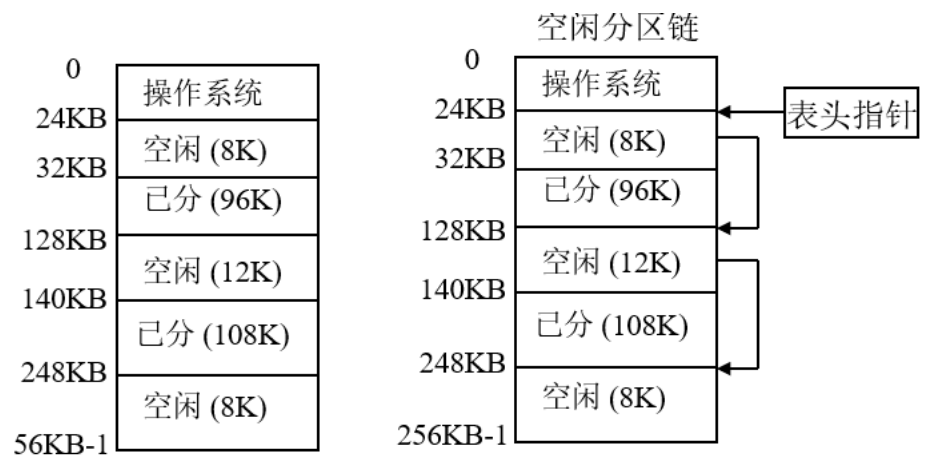
```

2.4 实验总结

- 动态分区：根据需要动态建立分区
 - 数据结构
 - ◆ 空闲分区表。用一个空闲分区表来登记系统中的空闲分区。其表项类似于固定分区。



- - ◆ 空闲分区链。将内存中的空闲分区以链表方式链接起来，构成空闲分区链。



内存布局图

- 首次适应算法：空闲分区按地址递增次序排列，找到第一个满足大小的空闲分区，划分一块分配，剩余空间留在空闲分区表中
 - 特点：优先利用内存低地址端，高地址端有大空闲区。但低地址端有许多小空闲分区时会增加查找开销。

3 磁盘存储空间的分配和回收

3.1 实验内容及上机实验所用平台

3.1.1 实验内容

模拟磁盘空闲空间的表示方法，以及模拟实现磁盘空间的分配和回收。

3.1.2 上机平台

系统: Windows

IDE: vscode

3.2 数据结构及代码段分析

一些常数: 柱面数、磁道数、记录数:

```
1. const int zhumian = 8, cidao = 2, jilu = 4;
2. #define total (zhumian*cidao*jilu)
```

将十进制转换为二进制的字符串:

```
3. string intToBiStr(int x, int len){
4.     string ret = "";
5.     while(x){
6.         ret += x%2+'0';
7.         x/=2;
8.     }
9.     while(ret.length() < len) ret+="0";
10.    reverse(ret.begin(), ret.end());
11.    return ret;
12. }
```

位置图的设置与读取:

```
13. namespace BITMAP{
14.     int bitmap[65];
```

1) 将二进制字符串转换为十进制整数

```
15.     int biToInt(const char* x){
16.         int len = strlen(x);
```



```

17.     int ret = 0;
18.     for(int i = 0; i < len; i++){
19.         if(x[i] == '1') ret += pow(2, len - i - 1);
20.     }
21.     return ret;
22. }

```

2) 获取位示图的某一位

```

23.     int getBit(int x){
24.         return bitmap[x];
25.     }
26.
27.     int getBit(const char* x){
28.         return bitmap[biToInt(x)];
29.     }

```

3) 设置位示图的某一位

```

30.     void setBit(int x, bool f){
31.         bitmap[x] = f;
32.     }
33.
34.     void setBit(char *s, bool f){
35.         bitmap[biToInt(s)] = f;
36.     }

```

4) 打印出整个位示图

```

37.     void displayBitMap(){
38.         cout<<"-----"<<endl;
39.
40.         for(int i = 0; i < zhumian; i++){
41.             cout<<i<<": ";
42.             for(int j = 0; j < cidao*jilu; j++)
43.                 cout<< getBit(i*cidao*jilu+j);
44.             cout<<endl;
45.         }
46.         cout<<"-----"<<endl;
47.     }
48. };

```

磁盘的相关操作:

```

49. namespace DISK{
50.     struct add{
51.         int zm,cd,jl;
52.     };

```

1) 申请空间

```

53.     using ret = pair<bool,add>;
54.     ret askForSpace(){
55.         int now = -1;
56.         for(int i = 0; i < total; i++){
57.             if(!BITMAP::getBit(i)){
58.                 now = i;
59.                 break;
60.             }
61.         }
62.         if(now >= 0){
63.             BITMAP::setBit(now, 1);
64.
65.             return make_pair(true, add{now/(cidoa*jilu), (now%(cidoa*jilu))/
66.                 4, (now%(cidoa*jilu))%4});
67.         }
68.         else {
69.             return make_pair(false, add{0,0,0});
70.         }
71.     }

```

2) 释放空间

```

72.     using ret2 = pair<bool, int>;
73.     ret2 releaseSpace(add a){
74.         int biAdd = a.zm*(cidoa*jilu) + a.cd*jilu + a.jl;
75.         if(!BITMAP::getBit(biAdd)){
76.             return make_pair(false, 0);
77.         }
78.         else {
79.             BITMAP::setBit(biAdd, 0);
80.             return make_pair(true, biAdd);
81.         }
82.     }
83. }
84.

```

Main 函数，初始化位示图和磁盘，接收来自键盘的指令并执行相关操作：

```

85. int n;
86. int main(){
87.     //freopen("h.in", "r", stdin);
88.     //cout<<intToBiStr(4, 4)<<endl;
89.     BITMAP::displayBitMap();
90.     // displayAvailTable();
91.     //return 0;
92.     cout << "how many opt?." << endl;
93.     cin >> n;
94.     for(int i = 1; i <= n; i++){
95.         cout <<endl<< "opt"<<i<<":"<<endl;
96.         char opt;
97.         cin >> opt;
98.         switch(opt){
99.             case 'A':{
100.                 DISK::ret ret = DISK::askForSpace();
101.                 if(ret.first){
102.                     cout << "succeeded, zhujian:"<<ret.second.zm<< " cidao:"
<<ret.second.cd<< " jilu:"<<ret.second.jl<<endl;
103.                 }
104.                 else cout << "failed, full." << endl;
105.                 break;
106.             }
107.             case 'R':{
108.                 int zm, cd, jl;
109.                 cin>>zm>>cd>>jl;
110.                 DISK::ret2 ret = DISK::releaseSpace(DISK::add{zm,cd,jl});
111.                 if(ret.first){
112.                     cout << "succeeded, bit address:"<<intToBiStr(ret.secon
d, 6)<<endl;
113.                 }
114.                 else cout<< "failed, it's available."<<endl;
115.                 break;
116.             }
117.         }
118.         BITMAP::displayBitMap();
119.     }
120.     return 0;
121. }

```

3.3 调试过程

进入 Windows/Linux 系统，输入 `g++ bitmap.cpp -o bitmap`
运行 `./bitmap`
运行结果如下：

```
1. PS D:\Study\大学资料\操作系统\实验\3\2> .\bitmap.exe
```

```
2. -----
```

```
3. 0: 00000000
```

```
4. 1: 00000000
```

```
5. 2: 00000000
```

```
6. 3: 00000000
```

```
7. 4: 00000000
```

```
8. 5: 00000000
```

```
9. 6: 00000000
```

```
10. 7: 00000000
```

```
11. -----
```

```
12. how many opt?.
```

```
13. 10
```

```
14.
```

```
15. opt1:
```

```
16. A
```

```
17. succeeded, zhumian:0 cidao:0 jilu:0
```

```
18. -----
```

```
19. 0: 10000000
```

```
20. 1: 00000000
```

```
21. 2: 00000000
```

```
22. 3: 00000000
```

```
23. 4: 00000000
```

```
24. 5: 00000000
```

```
25. 6: 00000000
```

```
26. 7: 00000000
```

```
27. -----
```

```
28.
```

```
29. opt2:
```

```
30. A
```

```
31. succeeded, zhumian:0 cidao:0 jilu:1
```

```
32. -----
```

```
33. 0: 11000000
```

```
34. 1: 00000000
```

```
35. 2: 00000000
```

```
36. 3: 00000000
```

```
37. 4: 00000000
```

```
38. 5: 00000000
```

```
39. 6: 00000000
```

```
40. 7: 00000000
```

```
41. -----
```

```
42.
43. opt3:
44. A
45. succeeded, zhumian:0 cidao:0 jilu:2
46. -----
47. 0: 11100000
48. 1: 00000000
49. 2: 00000000
50. 3: 00000000
51. 4: 00000000
52. 5: 00000000
53. 6: 00000000
54. 7: 00000000
55. -----
56.
57. opt4:
58. A
59. succeeded, zhumian:0 cidao:0 jilu:3
60. -----
61. 0: 11110000
62. 1: 00000000
63. 2: 00000000
64. 3: 00000000
65. 4: 00000000
66. 5: 00000000
67. 6: 00000000
68. 7: 00000000
69. -----
70.
71. opt5:
72. A
73. succeeded, zhumian:0 cidao:1 jilu:0
74. -----
75. 0: 11111000
76. 1: 00000000
77. 2: 00000000
78. 3: 00000000
79. 4: 00000000
80. 5: 00000000
81. 6: 00000000
82. 7: 00000000
83. -----
84.
85. opt6:
```

86. A
87. succeeded, zhumian:0 cidao:1 jilu:1
88. -----
89. 0: 11111100
90. 1: 00000000
91. 2: 00000000
92. 3: 00000000
93. 4: 00000000
94. 5: 00000000
95. 6: 00000000
96. 7: 00000000
97. -----
98.
99. opt7:
100. A
101. succeeded, zhumian:0 cidao:1 jilu:2
102. -----
103. 0: 11111110
104. 1: 00000000
105. 2: 00000000
106. 3: 00000000
107. 4: 00000000
108. 5: 00000000
109. 6: 00000000
110. 7: 00000000
111. -----
112.
113. opt8:
114. A
115. succeeded, zhumian:0 cidao:1 jilu:3
116. -----
117. 0: 11111111
118. 1: 00000000
119. 2: 00000000
120. 3: 00000000
121. 4: 00000000
122. 5: 00000000
123. 6: 00000000
124. 7: 00000000
125. -----
126.
127. opt9:
128. A
129. succeeded, zhumian:1 cidao:0 jilu:0

```

130. -----
131. 0: 11111111
132. 1: 10000000
133. 2: 00000000
134. 3: 00000000
135. 4: 00000000
136. 5: 00000000
137. 6: 00000000
138. 7: 00000000
139. -----
140.
141. opt10:
142. R 0 1 3
143. succeeded, bit address:000111
144. -----
145. 0: 11111110
146. 1: 10000000
147. 2: 00000000
148. 3: 00000000
149. 4: 00000000
150. 5: 00000000
151. 6: 00000000
152. 7: 00000000
153. -----

```

3.4 实验总结

- 磁盘上的一系列同心圆称为磁道（track），磁道沿径向又分成大小相等的多个扇区（sector），与盘片中心有一定距离的所有磁道组成一个柱面（cylinder）。
- 磁盘上的每个物理块可用柱面号，磁头号 and 扇区号表示。
- 逻辑扇区号=柱面号*每柱面扇区数+
 - 磁头号*每磁头扇区数+
 - 扇区号
- 位示图
 - 因位示图比较小，可以保存在主存中，因此空间的分配与回收较快；但需要进行位示图中二进制所在位置与盘块号之间的转换

4 进程创建

4.1 实验内容及上机实验所用平台

4.1.1 实验内容

利用 `fork()` 系统调用创建进程。

4.1.2 上机平台

系统: Windows + Linux 虚拟机

IDE: vscode

4.2 数据结构及代码段分析

```
1. #include<unistd.h>
2. #include<stdio.h>
3. int main()
4. {
5.     pid_t pid;
6.     int n; //count of child processes to create;
7.     char outputContent;
8.
9.     scanf("%d", &n);
10.    for(int i = 0; i < n; i++){
11.        pid=fork();
12.        if(pid== -1){
13.            printf("creation failed. error.\n");
14.            break;
15.        }
16.        else if(pid == 0){
17.            outputContent = i + 'A' ;
18.            printf("This is child process (id:%d) (parent process id:%d): %c\n", getpid(), getppid(), outputContent);
19.            break;
20.        }
21.        else {
22.            printf("This is parent process (id:%d), creating child Process %d\n", getpid(), pid);
23.        }
```



```
24.     }  
25.     return 0;  
26. }
```

fork() 函数用于复制父进程以生成一个子进程，它在父进程和子进程中都会有返回值：

- 1) -1: 创建子进程时出现错误。
- 2) 0: 创建子进程成功，当前进程为子进程。
- 3) >0: 创建子进程成功，当前进程为父进程。

创建多个进程时，如果 fork() 返回值为 0（即当前进程为子进程），那么就要退出循环，避免子进程继续创建子进程。

4.3 调试过程

进入 Ubuntu 虚拟机，输入 `gcc fork_ex.cpp -o fork_ex`

然后运行 `./fork_ex`

运行结果如下：

```
czyol@ubuntu:~/Documents/sysExperiment4_fork$ ./fork_ex  
2  
This is parent process (id:4408), creating child Process 4413  
This is child process (id:4413) (parent process id:4408): A  
This is parent process (id:4408), creating child Process 4414  
This is child process (id:4414) (parent process id:4408): B
```

4.4 实验总结

进程是资源分配的基本单位，也是独立运行的基本单位

4.4.1 引入进程的目的

- 1) 使多道程序能并发执行，以改善资源利用率及提高系统吞吐量
- 2) 描述程序动态执行过程的性质

4.4.2 进程和程序的区别

1) 进程是动态概念，程序是静态概念；进程是程序在处理机上的一次执行过程，而程序是指令的集合。

2) 进程是暂时的，程序是永久的。进程是一个状态变化的过程；程序可以长久保存。

3) 进程与程序的组成不同。进程的组成包括程序、数据和进程控制块。

- 4) 进程与程序是密切相关的。一个程序可以对应多个进程；一个进程可以包括多个程序。
- 5) 进程可以创建新进程，而程序不能形成新程序。

5 进程同步

5.1 实验内容及上机实验所用平台

5.1.1 实验内容

模拟实现同步机构，以避免发生进程执行时可能出现的与时间有关的错误。

5.1.2 上机平台

系统：Windows + Linux 虚拟机

IDE：vscode

5.2 数据结构及代码段分析

进程的状态：执行、就绪、等待、完成：

```
1. enum PROCESS_STATUS{  
2.     RUNNING, READY, WAITING, ZOMBIE  
3. };
```

每个进程都有一个 PCB 进程控制块，包含内容

进程名
状态
等待原因
断点

```
4. struct PCB{  
5.     std::string name;  
6.     PROCESS_STATUS status;  
7.     std::string reasonForWaiting;  
8.     int breakpoint;  
9.     int pc;  
10. };
```

信号量 S1、S2：

```
11. int s1, s2;
```

就绪队列，其中保存的是进程控制块的指针：

```
12. std::vector<PCB*> readyQueue;
```

产品池，生产者的产品将放在里面，消费者也将从中获取产品来消费：

```
13. char productPool[100];
14.
```

生产者和消费者的进程控制块：

```
15. PCB consumer, producer;
```

当前进程指针，指向生产者或消费者：

```
16. PCB *currentPCB;
```

临界区的相关操作：

```
17. namespace Sync{
```

1) P 操作

```
18. bool p(PCB &pcb, int &sem, std::string semName){
19.     sem--;
20.     if(sem < 0){
21.         pcb.status = WAITING;
22.         pcb.reasonForWaiting = semName;
23.         //pcb.reasonForWaiting.append(semName);
24.         printf("reason for waiting: %s\n", pcb.reasonForWaiting.c_str())
        ;
25.         pcb.breakpoint = pcb.pc;
26.         if(pcb.name == "consumer" && producer.status == ZOMBIE){
27.             currentPCB = nullptr;
28.             return false;
29.         }
30.         int randid = rand() % readyQueue.size();
31.         std::cout<<randid<<std::endl;
32.         currentPCB = readyQueue[randid];
33.         //std::cout<<randid<<" "<<currentPCB->name.c_str()<<" "<<current
        PCB->pc<<" "<<std::endl;
34.         readyQueue.erase(readyQueue.begin() + randid);
35.         return false;
36.
37.     }
38.     return true;
39. }
40.
```

2) V 操作

```
41.     bool v(PCB &pcb, int &sem, std::string semName){
42.         sem++;
43.         if(sem <= 0){
44.             //release a process
45.             std::string reasonForWaiting = semName;
46.             if(consumer.reasonForWaiting == semName) {
47.
48.                 consumer.status = READY;
49.                 consumer.reasonForWaiting = "";
50.                 consumer.pc = consumer.breakpoint;
51.                 readyQueue.push_back(&consumer);
52.                 printf("consumer is waken up\n");
53.             }
54.             else if(producer.reasonForWaiting == semName){
55.                 producer.status = READY;
56.                 producer.reasonForWaiting = "";
57.                 producer.pc = producer.breakpoint;
58.                 readyQueue.push_back(&producer);
59.                 printf("producer is waken up\n");
60.             }
61.
62.             return false;
63.         }
64.         return true;
65.     }
66. }
67.
```

生产者的相关操作:

```
68. namespace Producer{
69.     int id = 0;
70.     int in = 0;
71.     char C;
```

- 1) 生产者的指令集合, 为了模拟 PC 跳转, 将每一个 PC 值对应要执行的代码封装为一个函数; PA 中保存的是 pc 值到对应函数指针的映射:

```
72.     typedef void (*code)();
73.     std::map<int, code> PA;
```

- 2) 生产者执行完一条指令后, 进入就绪状态, 加入就绪队列。然后从就绪队列中随机选择一个进程作为当前进程:

```

74.     void ready(){
75.         producer.status = READY;
76.
77.         readyQueue.push_back(&producer);
78.         int randid = rand() % readyQueue.size();
79.         std::cout<<randid<<std::endl;
80.         currentPCB = readyQueue[randid];
81.         readyQueue.erase(readyQueue.begin() + randid);
82.         printf("already put into ready queue\n");
83.     }

```

3) 生产者生产产品

```

84.     void produce(){
85.         C = 'A' + (id++);
86.         printf("producing a product %c\n", C);
87.         ready();
88.     }

```

4) 生产者将生产的产品放入产品池中

```

89.     void put(){
90.         productPool[in++] = C;
91.         in %= 10;
92.         printf("putting product %c into the pool\n", C);
93.         ready();
94.     }

```

5) 生产者执行 P(s1)操作

```

95.     void ps1(){
96.         printf("producer is excuting p(s1) %d\n", s1 - 1);
97.         if(Sync::p(producer,s1, "s1")){
98.             ready();
99.         }else{
100.             printf("producer is waiting\n");
101.         }
102.     }

```

6) 生产者执行 V(s2)操作

```

103.    void vs2(){
104.        printf("producer is excuting v(s2) %d\n", s2 + 1);
105.        if(Sync::v(producer,s2, "s2"))
106.            ready();

```

```

107.         else {
108.             //printf("producer is waken up\n");
109.         }
110.     }

```

7) 生产者执行 goto(0)操作，回到第一条语句：

```

111.     void goto0(){
112.         printf("producer is excuting goto(0)\n");
113.         producer.pc = 0;
114.         ready();
115.     }
116. }

```

消费者的相关操作，与生产者类似：

```

117. namespace Consumer{
118.     int outId = 0;
119.     char C;
120.     typedef void (*code)();
121.     std::map<int, code> SA;
122.
123.     void ready(){
124.         consumer.status = READY;
125.         readyQueue.push_back(&consumer);
126.         int randid = rand() % readyQueue.size();
127.         std::cout<<randid<<std::endl;
128.         currentPCB = readyQueue[randid];
129.         readyQueue.erase(readyQueue.begin() + randid);
130.         printf("already put into ready queue\n");
131.     }
132.
133.     void get(){
134.         C = productPool[outId++];
135.         printf("consumer got a product %c from the pool\n", C);
136.         outId %= 10;
137.         ready();
138.     }
139.     void consume(){
140.         printf("consumer is consuming product %c\n", C);
141.         int ip = ((outId - 1) + 10) % 10;
142.         productPool[ip] = '0';
143.         ready();
144.     }
145.     void ps2(){

```

```

146.     printf("consumer is excuting p(s2) %d\n", s2 - 1);
147.     if(Sync::p(consumer,s2, "s2"))
148.         ready();
149.     else printf("consumer is waiting\n");
150. }
151. void vs1(){
152.     printf("consumer is excuting v(s1) %d\n", s1 + 1);
153.     if(Sync::v(consumer,s1, "s1"))
154.         ready();
155.     else{
156.         //printf("consumere is waken up\n");
157.     }
158. }
159. void goto0(){
160.     printf("consumer is excuting goto(0)\n");
161.     consumer.pc = 0;
162.     ready();
163. }
164. }

```

初始化消费者进程，包括名字、状态、等待原因、pc 值、每个 pc 值所对应的操作（函数指针）：

```

165. void initConsumer(){
166.     consumer.name = "consumer";
167.     consumer.status = READY;
168.     consumer.reasonForWaiting = "";
169.     consumer.breakpoint = 0;
170.     consumer.pc = 0;
171.     Consumer::SA[0] = Consumer::ps2;
172.     Consumer::SA[1] = Consumer::get;
173.     Consumer::SA[2] = Consumer::vs1;
174.     Consumer::SA[3] = Consumer::consume;
175.     Consumer::SA[4] = Consumer::goto0;
176. }

```

初始化生产者进程，包括名字、状态、等待原因、pc 值、每个 pc 值所对应的操作（函数指针）：

```

177. void initProducer(){
178.     producer.name = "producer";
179.     producer.status = READY;
180.     producer.reasonForWaiting = "";
181.     producer.breakpoint = 0;
182.     producer.pc = 0;

```



```

183.     Producer::PA[0] = Producer::produce;
184.     Producer::PA[1] = Producer::ps1;
185.     Producer::PA[2] = Producer::put;
186.     Producer::PA[3] = Producer::vs2;
187.     Producer::PA[4] = Producer::goto0;
188. }

```

Main 函数，接收来自键盘的命令并执行相应操作：

```

189. int main(){
190.     initConsumer();
191.     initProducer();
192.
193.     s1 = 10;
194.     s2 = 0;
195.
196.     srand(0);
197.     readyQueue.push_back(&consumer);
198.     currentPCB = &producer;
199.     std::cout<<"starting...\n"<<std::endl;
200.     while(currentPCB != nullptr){
201.         printf("-----\n");
202.         printf("\n\npool: ");
203.         for(int i=0; i < Producer::in; i++){
204.             printf("%c ", productPool[i]);
205.         }
206.         printf("\nreadyQueue:");
207.         for(auto x : readyQueue){
208.             printf("%s ", x->name.c_str());
209.         }
210.         printf("\n");
211.         printf("currently running: %s, pc:%d\n", currentPCB->name.c_str(),
            currentPCB->pc);
212.         //char x; std::cin>>x;
213.         int curPC = (*currentPCB).pc++;
214.         if(currentPCB->name == "producer"){
215.             (*Producer::PA[curPC])();
216.             char opt;
217.             printf("stop the producer from running? y/n ");
218.             std::cin >> opt;
219.             if(opt == 'y' || opt == 'Y'){
220.                 producer.status = ZOMBIE;
221.                 currentPCB = &consumer;
222.                 readyQueue.clear();
223.             }

```

```

224.         else if(opt == 'n' || opt == 'N'){
225.             continue;
226.         }
227.     }
228.     else if(currentPCB->name == "consumer"){
229.         (*Consumer::SA[curPC])();
230.         char opt;
231.         std::cin >> opt;
232.     }
233. }
234.
235. }

```

5.3 调试过程

进入 Ubuntu 虚拟机，输入 `gcc fork_ex.cpp -o fork_ex`
 然后运行 `./fork_ex`
 运行结果如下：

```

1. czyol@ubuntu:~/Documents/sysExperiment5_pv_sem$ ./pv
2. starting...
3.
4. -----
5.
6.
7. pool:
8. readyQueue:consumer
9. currently running: producer, pc:0
10. producing a product A
11. 1
12. already put into ready queue
13. stop the producer from running? y/n n
14. -----
15.
16.
17. pool:
18. readyQueue:consumer
19. currently running: producer, pc:1
20. producer is excuting p(s1) 9
21. 0
22. already put into ready queue
23. stop the producer from running? y/n n

```

```
24. -----
25.
26.
27. pool:
28. readyQueue:producer
29. currently running: consumer, pc:0
30. consumer is excuting p(s2) -1
31. reason for waiting: s2
32. 0
33. consumer is waiting
34.
35. .
36. -----
37.
38.
39. pool:
40. readyQueue:
41. currently running: producer, pc:2
42. putting product A into the pool
43. 0
44. already put into ready queue
45. stop the producer from running? y/n n
46. -----
47.
48.
49. pool: A
50. readyQueue:
51. currently running: producer, pc:3
52. producer is excuting v(s2) 0
53. consumer is waken up
54. stop the producer from running? y/n n
55. -----
56.
57.
58. pool: A
59. readyQueue:consumer
60. currently running: producer, pc:4
61. producer is excuting goto(0)
62. 1
63. already put into ready queue
64. stop the producer from running? y/n n
65. -----
66.
67.
```

```
68. pool: A
69. readyQueue:consumer
70. currently running: producer, pc:0
71. producing a product B
72. 1
73. already put into ready queue
74. stop the producer from running? y/n n
75. -----
76.
77.
78. pool: A
79. readyQueue:consumer
80. currently running: producer, pc:1
81. producer is excuting p(s1) 8
82. 0
83. already put into ready queue
84. stop the producer from running? y/n n
85. -----
86.
87.
88. pool: A
89. readyQueue:producer
90. currently running: consumer, pc:1
91. consumer got a product A from the pool
92. 0
93. already put into ready queue
94. .
95. -----
96.
97.
98. pool: A
99. readyQueue:consumer
100. currently running: producer, pc:2
101. putting product B into the pool
102. 1
103. already put into ready queue
104. stop the producer from running? y/n n
105. -----
106.
107.
108. pool: A B
109. readyQueue:consumer
110. currently running: producer, pc:3
111. producer is excuting v(s2) 1
```

```
112. 1
113. already put into ready queue
114. stop the producer from running? y/n y
115. -----
116.
117.
118. pool: A B
119. readyQueue:
120. currently running: consumer, pc:2
121. consumer is excuting v(s1) 9
122. 0
123. already put into ready queue
124. .
125. -----
126.
127.
128. pool: A B
129. readyQueue:
130. currently running: consumer, pc:3
131. consumer is consuming product A
132. 0
133. already put into ready queue
134. .
135. -----
136.
137.
138. pool: 0 B
139. readyQueue:
140. currently running: consumer, pc:4
141. consumer is excuting goto(0)
142. 0
143. already put into ready queue
144. .
145. -----
146.
147.
148. pool: 0 B
149. readyQueue:
150. currently running: consumer, pc:0
151. consumer is excuting p(s2) 0
152. 0
153. already put into ready queue
154. .
155. -----
```

```
156.
157.
158. pool: 0 B
159. readyQueue:
160. currently running: consumer, pc:1
161. consumer got a product B from the pool
162. 0
163. already put into ready queue
164. .
165. -----
166.
167.
168. pool: 0 B
169. readyQueue:
170. currently running: consumer, pc:2
171. consumer is excuting v(s1) 10
172. 0
173. already put into ready queue
174. .
175. -----
176.
177.
178. pool: 0 B
179. readyQueue:
180. currently running: consumer, pc:3
181. consumer is consuming product B
182. 0
183. already put into ready queue
184. .
185. -----
186.
187.
188. pool: 0 0
189. readyQueue:
190. currently running: consumer, pc:4
191. consumer is excuting goto(0)
192. 0
193. already put into ready queue
194. .
195. -----
196.
197.
198. pool: 0 0
199. readyQueue:
```

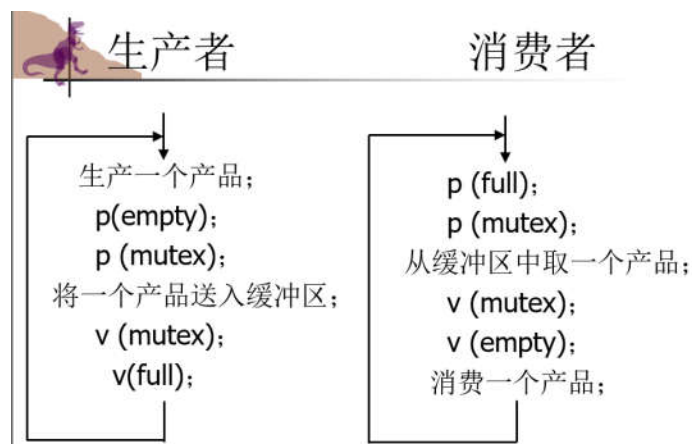
```

200. currently running: consumer, pc:0
201. consumer is excuting p(s2) -1
202. reason for waiting: s2
203. consumer is waiting
204. .

```

5.4 实验总结

- 生产者-消费者：缓冲池中的每个缓冲区可以存放一个产品，生产者进程不断生产产品并将产品放入缓冲池中，消费者进程不断从缓冲池内取出产品并消费。
 - 设置两个同步信号量 `empty`、`full`，其初值分别为 `n`、`0`。
 - 有界缓冲池是一个临界资源，还需要设置一个互斥信号量 `mutex`，其初值为 `1`。



- 两者的两个 `p` 操作均不能颠倒，否则可能造成死锁

参考文献

教师评语评分

评语： _____

评分： _____

评阅人：

年 月 日

（备注：对该实验报告给予优点和不足的评价，并给出百分之评分。）