

**操作系统原理课程实验报告**

|  |  |
| --- | --- |
| 姓 名： | 汪宇飞 |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS2003班 |
| 学 号： | U202015375 |
| 指导教师： | 谢美意 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2023年03月30日

目 录

[实验一 lab3\_challenge2：实现信号量 1](#_Toc123244734)

[1.1 实验目的 1](#_Toc123244735)

[1.2 实验内容 1](#_Toc123244736)

[1.3 实验调试及心得 6](#_Toc123244737)

# 实验一 lab3\_challenge2：实现信号量

## 实验目的

给定应用app\_semaphore.c，通过信号量的增减，控制主进程和两个子进程的输出按主进程，第一个子进程，第二个子进程，主进程，第一个子进程，第二个子进程……这样的顺序轮流输出。实验的要求是，通过修改PKE内核和系统调用，为用户程序提供信号量功能

通过阅读并分析应用app\_semaphore.c中的函数可以看出，函数里调用了sem\_new()、sem\_P()和sem\_V()三个未定义的函数，通过它们的函数名并分析它们在应用中的作用可以得知，这三个函数分别实现的功能是信号量的分配、信号量的P操作和信号量的V操作。

## 实验内容

由上述实验目的的讲述与分析可得出实验内容分为两个部分：一是需要定义信号量的结构体，结构体需要包括合理的变量使得信号量的处理能够顺利进行；二是需要依次实现上述三个未定义的函数，使它们能够正确地处理信号量的分配和P、V操作。

对于第一个内容，根据信号量本身的功能和后续函数的实现，设计了如下信号量结构体sem\_t。

其中，整型变量val为信号量的值，整型变量occupied为1时表示当前该信号量已被使用，为0时表示未被使用，进程指针h和t分别为等待该信号量的进程的队列的头和尾的指针。

1. typedef struct {
2. int val, occupied;
3. process \*h, \*t;
4. } sem\_t;

其次对于第二个内容，首先需要添加对应的函数调用，需要在user\_lib.h文件中添加sem\_new()，sem\_P()和sem\_V()函数原型，并在user\_lib.c文件中添加sem\_new()，sem\_P()和sem\_V()函数实现。这三个函数均接收一个int变量作为参数，返回一个int值，其具体实现可以参照user\_lib.c文件中其他函数的实现，调用并返回do\_user\_call函数的返回值即可，对于参数值a0传入对应函数的ID号，a1传入对应函数的整型变量，剩下六个参数均设置为0。如代码1.1所示。

**代码1.1 user\_lib部分的修改**

1. *//以下为位于user\_lib.h文件中的三个函数原型*
2. int sem\_new(int);
3. int sem\_P(int);
4. int sem\_V(int);
5. *//以下为位于user\_lib.c文件中的三个函数实现*
6. int sem\_new(int value) {
7. return do\_user\_call(SYS\_sem\_new, value, 0, 0, 0, 0, 0, 0);
8. }
9. int sem\_P(int sem) {
10. return do\_user\_call(SYS\_sem\_P, sem, 0, 0, 0, 0, 0, 0);
11. }
12. int sem\_V(int sem) {
13. return do\_user\_call(SYS\_sem\_V, sem, 0, 0, 0, 0, 0, 0);
14. }

在声明和定义上述三个函数之后，需要对于syscall.h和syscall.c进行修改使得do\_user\_call函数可以对三个函数相关操作进行相应与实现。首先需要在syscall.h文件中对于SYS\_sem\_new、SYS\_sem\_P和SYS\_sem\_V进行定义，使得do\_user\_call函数可以正确识别并处理。之后需要在syscall.c文件中对do\_syscall函数进行修改，在switch语句中添加对应的三个case，并增加对应函数进行处理，该函数接收对应的整型变量作为参数，调用对应的内核函数newsem()、p()和v()并返回这些内核函数的返回值，如代码1.2所示。

**代码1.2 syscall部分的修改**

1. *//以下为位于syscall.h文件中的新增定义*
2. #define SYS\_sem\_new (SYS\_user\_base + 6)
3. #define SYS\_sem\_P (SYS\_user\_base + 7)
4. #define SYS\_sem\_V (SYS\_user\_base + 8)
5. *//以下为位于syscall.c文件中的函数实现与修改*
6. ssize\_t sys\_sem\_new(int value) {
7. return newsem(value);
8. }
9. ssize\_t sys\_sem\_P(uint64 sem) {
10. return p(sem);
11. }
12. ssize\_t sys\_sem\_V(uint64 sem) {
13. return v(sem);
14. }
15. long do\_syscall(long a0, long a1, long a2, long a3, long a4, long a5, long a6, long a7) {
16. switch (a0) {
17. case SYS\_user\_print:
18. return sys\_user\_print((const char\*)a1, a2);
19. case SYS\_user\_exit:
20. return sys\_user\_exit(a1);
21. case SYS\_user\_allocate\_page:
22. return sys\_user\_allocate\_page();
23. case SYS\_user\_free\_page:
24. return sys\_user\_free\_page(a1);
25. case SYS\_user\_fork:
26. return sys\_user\_fork();
27. case SYS\_user\_yield:
28. return sys\_user\_yield();
29. *//以下为新增的三个case*
30. case SYS\_sem\_new:
31. return sys\_sem\_new(a1);
32. case SYS\_sem\_P:
33. return sys\_sem\_P(a1);
34. case SYS\_sem\_V:
35. return sys\_sem\_V(a1);
36. default:
37. panic("Unknown syscall %ld \n", a0);
38. }
39. }

在修改完syscall部分后，需要新增内核函数newsem()、p()和v()。首先需要在process.h文件中添加函数原型，并在process.c文件中添加函数实现，以下分别给出上述三个内核函数的代码和分析。

1. newsem函数

newsem函数的功能是实现对信号量的分配。

代码定义了一个包含NPROC个信号量sem\_t结构体的数组sems。函数newsem将信号量初始化为value的值，函数遍历sems数组，找到第一个未被占用的信号量并将其初始化，将其val值设置为value值，occupied置为1，然后返回该信号在数组sems中的下标。如果数组中全部的信号量都被占用了，则返回-1表示创建新信号量失败。函数如代码1.3所示。

**代码1.3 newsem函数**

1. *//以下为位于process.h文件中的函数原型*
2. int newsem(int);
3. *//以下为位于process.c文件中的do\_wait函数实现*
4. sem\_t sems[NPROC];
5. int newsem(int value) {
6. int i = 0;
7. sem\_t \*sem = NULL;
8. while (i < NPROC) {
9. sem = &sems[i];
10. if (!sem->occupied) {
11. sem->occupied = 1;
12. sem->val = value;
13. sem->h = sem->t = NULL;
14. return i;
15. }
16. i++;
17. }
18. return -1;
19. }

2. p函数

p函数的功能是实现对信号量的P操作。

首先函数对给定的变量也即传入的信号量编号sem进行有效性检查，如果该编号不在有效范围内则返回-1表示操作失败。然后函数获取信号量在sems数组中的指针，将其信号量值减1。如果值小于0表示没有可用资源了，需要通过修改信号量的头尾指针h和t将其加入阻塞队列，并调用schedule()函数进行调度。进程再次被调度执行的时候会被从阻塞队列中取出回到主循环。函数如代码1.4所示。

**代码1.4 p函数**

1. *//以下为位于process.h文件中的函数原型*
2. int p(int);
3. *//以下为位于process.c文件中的do\_wait函数实现*
4. int p(int sem) {
5. if (sem < 0 || sem >= NPROC) {return -1;}
6. sem\_t \*semaphore = &sems[sem];
7. semaphore->val--;
8. if (semaphore->val < 0) {
9. process \*ti = current;
10. if (semaphore->h == NULL) {
11. semaphore->h = semaphore->t = ti;
12. ti->queue\_next = NULL;
13. } else {
14. semaphore->t->queue\_next = ti;
15. semaphore->t = ti;
16. ti->queue\_next = NULL;
17. }
18. ti->status = BLOCKED;
19. schedule();
20. }
21. return 0;
22. }

3. v函数

v函数的功能是实现对信号量的V操作。

首先v函数与p函数相同，对传入的信号量编号sem进行有效性检查。然后函数获取信号量在sems数组中的指针，将其信号量值加1。如果值大于等于0表示有进程在等待该信号量，所以需要唤醒其中一个进程，函数通过修改信号量的头尾指针h和t从该信号量的阻塞队列中取出一个进程并将其插入就绪队列中。函数如代码1.5所示。

**代码1.5 v函数**

1. *//以下为位于process.h文件中的函数原型*
2. int v(int);
3. *//以下为位于process.c文件中的do\_wait函数实现*
4. int v(int sem) {
5. if (sem < 0 || sem >= NPROC) {return -1;}
6. sem\_t \*semaphore = &sems[sem];
7. semaphore->val++;
8. if (semaphore->h != NULL) {
9. process \*t = semaphore->h;
10. semaphore->h = t->queue\_next;
11. if (semaphore->h == NULL) {semaphore->t = NULL;}
12. insert\_to\_ready\_queue(t);
13. }
14. return 0;
15. }

至此，完成了对于PKE内核和系统调用的修改，完成了对sem\_new()、sem\_P()和sem\_V()三个未定义函数的实现，使所给定应用的app\_semaphore.c能够按照预期运行并输出。

## 实验调试及心得

本次挑战实验lab3\_challenge2是最后一个挑战实验，与我预期相反，它反而是操作系统课设的四个挑战实验中相对比较简单的一个了。它要求实现的信号量的分配、P、V操作都是在课堂上老师详细讲过的，有了扎实的理论基础在实现代码的时候更加得心应手。

话虽如此，毕竟课设距离实验过去了一个多月，需要再次重新阅读指导。我花费了较多时间回过头来重新阅读基础实验的指导并进行理解，诸如系统调用与内核之间的联系、syscall的实现等，都是需要从头再阅读并理解一次的。在实现期间也多次使用educoder所提供的命令行进行测试，让我再一次感谢老师为我们提供了这一方便的实验环境。在经历多次报错之后能够成功实现并通关还是十分有成就感的。

这次实验是我所做过的最具特色的实验之一，9个基础实验和6个挑战实验，以十分详尽的讲解为我们介绍了操作系统底层的组成与结构，让人不禁感叹计算机操作系统的精致巧妙以及严谨性，为我们展示了理论课之外更加细致具体的部分。十分感谢课程组的老师们对于本次实验的设计以及实验指导的撰写和实验平台和资料的部署，为我们提供了一个方便的实验环境和详尽的指导。