华中科技大学

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

1.1 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。

1.2 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

```
//以下为位于user_lib.h 文件中的三个函数原型
int sem_new(int);
int sem_P(int);
int sem_V(int);

//以下为位于user_lib.c 文件中的三个函数实现
int sem_new(int value) {
    return do_user_call(SYS_sem_new, value, 0, 0, 0, 0, 0, 0);
}

int sem_P(int sem) {
    return do_user_call(SYS_sem_P, sem, 0, 0, 0, 0, 0, 0);
}

int sem_V(int sem) {
    return do_user_call(SYS_sem_V, sem, 0, 0, 0, 0, 0, 0);
}
```

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

```
//以下为位于syscall.h 文件中的新增定义
#define SYS_sem_new (SYS_user_base + 6)
#define SYS_sem_P (SYS_user_base + 7)
#define SYS_sem_V (SYS_user_base + 8)

//以下为位于syscall.c 文件中的函数实现与修改
ssize_t sys_sem_new(int value) {
    return newsem(value);
}

ssize_t sys_sem_P(uint64 sem) {
    return p(sem);
```

```
13.}
14.
15.ssize_t sys_sem_V(uint64 sem) {
16.    return v(sem);
17.}
18.
19.long do_syscall(long a0, long a1, long a2, long a3, long a4, long
    a5, long a6, long a7) {
20.  switch (a0) {
21.    case SYS_user_print:
22.      return sys_user_print((const char*)a1, a2);
23.    case SYS_user_exit:
24.      return sys_user_exit(a1);
25.    case SYS_user_allocate_page:
26.      return sys_user_allocate_page();
27.    case SYS_user_free_page:
28.      return sys_user_free_page(a1);
29.    case SYS_user_fork:
30.      return sys_user_fork();
31.    case SYS_user_yield:
32.      return sys_user_yield();
33.//以下为新增的三个case
34.    case SYS_sem_new:
35.      return sys_sem_new(a1);
36.    case SYS_sem_P:
37.      return sys_sem_P(a1);
38.    case SYS_sem_V:
39.      return sys_sem_V(a1);
40.    default:
41.      panic("Unknown syscall %ld \n", a0);
42.  }
43.}
```

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.
4. //以下为位于process.c 文件中的do_wait 函数实现
5. sem_t sems[NPROC];
6. int newsem(int value) {
7.   int i = 0;
8.   sem_t *sem = NULL;
9.   while (i < NPROC) {
10.    sem = &sems[i];
11.    if (!sem->occupied) {
12.      sem->occupied = 1;
13.      sem->val = value;
14.      sem->h = sem->t = NULL;
15.      return i;
16.    }
17.    i++;
18.  }
19.  return -1;
20.}
```

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.
4. //以下为位于process.c 文件中的do_wait 函数实现
5. int p(int sem) {
6.   if (sem < 0 || sem >= NPROC) {return -1;}
7.   sem_t *semaphore = &sems[sem];
8.   semaphore->val--;
```

```
9.    if (semaphore->val < 0) {
10.     process *ti = current;
11.     if (semaphore->h == NULL) {
12.       semaphore->h = semaphore->t = ti;
13.       ti->queue_next = NULL;
14.     } else {
15.       semaphore->t->queue_next = ti;
16.       semaphore->t = ti;
17.       ti->queue_next = NULL;
18.     }
19.     ti->status = BLOCKED;
20.     schedule();
21.   }
22.   return 0;
23. }
```

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.
4. //以下为位于process.c 文件中的do_wait 函数实现
5. int v(int sem) {
6.   if (sem < 0 || sem >= NPROC) {return -1;}
7.   sem_t *semaphore = &sems[sem];
8.   semaphore->val++;
9.   if (semaphore->h != NULL) {
10.    process *t = semaphore->h;
11.    semaphore->h = t->queue_next;
12.    if (semaphore->h == NULL) {semaphore->t = NULL;}
13.    insert_to_ready_queue(t);
14.  }
15.  return 0;
16.}
```

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。

1.3 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。