Lab6 实验报告

习题二

`sleep` 函数主要需更改进程运行状态到 *SLEEPING* ，同时记录该进程所等待的信号，之后调用 `sched` 函数调度一个可执行进程在当前 CPU 上执行。在程序睡眠的过程中，需要释放掉进程所持有的资源，以供其它进程在改进程睡眠时使用；当进程结束睡眠，返回调用程序继续执行时，需要让调用程序认为它仍然占用着资源。故在进入 `sleep` 时会先释放掉进程持有的锁，在退出时再上锁。

`wakeup` 函数需要找到等待对应信号的进程，更改该进程的状态。

sd 卡实现

sd 卡的读写请求是一个通过链表来维护的队列。当收到 sd 卡读写完成的中断信号时，系统进入 `sd_intr` 函数完成相关的数据读取，之后调用 `wakeup` 函数唤醒等待该读写完成信号的进程。接着，再从队列中找到下一个请求发送给 sd 卡。当进程需要向 sd 卡发送请求时，只需向该队列的末尾添加请求并调用 `sleep` 将进程睡眠等待读写完成信号。使用了生产者——消费者模型，该队列就可充当生产者和消费者之间的通信管道，通过锁机制来保证该通道的一致性。

`sd_init` 函数负责对 sd 卡进行初始化。在初始化队列和锁之后，`sd_init` 还需要读取 sd 卡固定位置的数据。该读取操作使用 `sd_start` 函数完成。需要注意在开始从 sd 卡缓冲中读取数据前会以 *INT_READ_RDY* 和 *INT_DATA_DONE* 中断信号作为标志，需调用 `sdWaitForInterrupt` 函数等待这些信号并对其屏蔽。

`sd_rw` 函数负责处理 sd 卡读写请求。如果处理队列为空，就调用 `sd_start` 向 sd 卡发送请求，由于 `sd_intr` 中会将下一个请求通过 `sd_start` 发送出去，那么如果队列不为空，该入队的请求就可以在队列中前一个请求完成后被 `sd_intr` 发送，也就不用在 `sd_rw` 发送了。**注意：由于 `sd_intr` 中先申请了 *sdlock* ，之后在 `wakeup` 中申请 *ptablelock*；在 `sd_rw` 中先申请 *sdlock* 并执行 `sd_start` 发送请求，如果在调用 `sleeping` 时先释放 *sdlock*，如果此时 `sd_start` 的请求被完成，另外的 CPU 在进入 `sd_insr` 处理 sd 卡完成中断时可能会先进入 `wake_up` 函数，由于此时 *ptablelock* 并未被 `sleep` 占**

用，运行在另外 CPU 上的 `wake_up` 就会先于 `sleep` 被执行，导致程序出错。

在 `sleep` 中需要先申请 *ptablelock* 再释放 *sdlock* !!!

读写性能

在 *initcode.S* 中添加 `while (1);` 语句，并更改 `syscall` 函数调用 `sd_test` 。

在执行中，一个进程负责执行 *inintcode* 调用 `syscall` 进入 `sd_test` 进行测试，多个空进程执行 `while(1);` 语句来等待中断。当出现 sd 卡中断时，空进程进入 `sd_intr` 处理 sd 卡对执行 `sd_test` 的进程发出请求的回应并将原本 *SLEEPING* 状态的测试进程唤醒；当出现时钟中断时，空进程进入 `yield` 函数重新进行调度，所在 CPU 开始执行原本睡眠的测试进程。

性能优化

- 空进程数：考虑到 *test* 中 sd 的读写需要 CPU 收到中断后才能进行下一步操作，那么一个想法便是让每次时钟中断处理器都能运行 *test* 而不会运行空进程。因此，过多的空进程会导致 CPU 在时钟中断时选择空进程而不进行下一步 *test*，而且会带来更多的调度开销；而过少的空进程会导致空闲的 CPU 一直执行调度程序保持在 *el1* 状态，无法响应时钟中断，也就无法通过 `yield` 切换到 *test*。最终选择了启用 4 个空进程以达到最优性能。
- 哈希寻找：考虑到 `wakeup` 中需要遍历所有的进程，查找等待对应信号的进程并唤醒。可以用哈希挂链的形式存储所有等待信号以及对应进程，当调用 `wakeup` 时利用遍历对应哈希值下的信号找到对应进程。
- 调度方法：本系统使用了 3 级优先级，每个优先级下的 *RUNNABLE* 进程都会用一个链表连接的队列来维护。如果某个进程频繁用光时间片，那么它就会被降低优先级。这样有利于 `yield` 时选择优先级高的进程 *test*，保证在进程较多时仍能优先选择 IO 密集型（sd_test）。为了重要的防止 CPU 密集型的进程一直处于低优先级，会将长时间处于低优先级的进程重新提到高优先级。