OPERATING SYSTEM

TD 2: Sleep and awake of processes

Global instructions

Suppose that the processes are completely defined by their proc structure containing the following fields:

```
struct proc {
    struct proc * p_next;    /* list link */
    int p_status;            /* status of the process */
    ...
}
```

- ✓ The system has access to the global `struct proc * current` variable pointing to an active process on the processor.
- ✓ A running process doesn't belong to the run queue `runq`: the scheduler removes it from `runq` when it chooses it.
- ✓ The `p_status` field is set to `STATUS_BLOCKED` or `STATUS_READY`.
- ✓ All `STATUS_READY` processes are in a global linked list `struct proc* runq`, linked by the `p_next` field.
- ✓ All `STATUS_BLOCKED` processes are in a global linked list `struct proc* sleepq`, also linked by the `p_next` field.
- ✓ We will start by ignore the process order in `runq` and `sleepq`.
- ✓ A `switch()` procedure asks the scheduler to save the current process context and to execute another process among the `STATUS_READY`.
- ✓ When the first process takes control, it executes the code just after the `switch()` instruction.

1 Basic implementation

(1.1) Write the functions `wait()` and `wakeup()` allowing to asleep and wake up the processes.

(1.2) The `wakeup()` function can be called during an interruption. Give an example. What are the problems? How resolve these problems if we have `irq_disable()` and `irq_enable()` functions allowing to hide and reactivate interruptions?

(1.3) Suppose that a process wants to fall asleep, and just before this moment, an interrupt awakens all waiting processes. Consider that interruptions during the masking are stored and delivered during their reactivation. What is going to happen? Is that bad? What solutions do you propose?

(1.4) If our machine was multiprocessor or multicore, what would you have to change in your code?

2 Receiving network packets

We now deal with the concrete case of a network reception. Processes fall asleep while awaiting a packet (for example, in the `recv()` system call) and are awakened by an interruption in the network interface controller when it received a packet.
We have a global variable `received_count` describing the number of packets received by the network interface controller and not yet consumed by a process. A process call the `consume()` function to consume a packet after checking that `received_count` is strictly positive. Each packet can only be consumed once.

(2.1) Specify your `wakeup()` and `wait()` codes to fall asleep only if no network packet is available, and verify that there is one available on waking up.

(2.2) Modify `received_count` to contain the number of received bytes instead of the number of packets. The `consume()` function will also take a number of bytes as parameter. Modify `wakeup()` to increment `received_count` from a new `length` parameter. Then, change `wait()` to consume as many bytes as indicated in a new `length` parameter.

(2.3) If multiple processes expect bytes at the same time and only one can run at a time, what are the weaknesses of this implementation? How to solve them?

(2.4) When would a process expect a network packet without consuming it immediately? How to take this into account?

(2.5) Your goal is to make distinction between connections in which the packets arrive. Modify the `wakeup()` and `wait()` functions to take into account the target connection that will be given in a new `struct conn * conn` argument.

(2.6) How improve your code in order to avoid browsing the entire `sleepq` if no process is waiting on the target socket?

3 Extensions

(3.1) Formalize the notion of an event that can be wait by a process, and precise what the associated structure must contain.

(3.2) How to manage the case where a process wants to sleep until an event among several occurs? In which case is it useful? How to allocate all the required structures? What if we want to waiting for all the events happen?

(3.3) How to improve your code to take into account a priority field in the `proc` structure? We want the priority processes be awakened first and executed first, so you must sort the processes in `runq` and `sleepq`. What could be the problem with this model when there are many processes with different priorities? How to solve it?

(3.4) What should be done if a signal arrives while a process is sleeping? All reasons for sleeping are they equivalent? Suggest a solution.

4 More concrete examples

(4.1) How to adapt the code if we consider a reading in a pipe instead of a network connection? How to block a process if it writes too much in the pipe?

(4.2) Implement a semaphore (`sem_init`, `sem_post`, `sem_wait`). Ensure that it can be used in an interrupt handler. Give a concrete example where this could happen.

(4.3) Implement condition variables (`pthread_cond_wait`, `signal` and `broadcast`). What are the similarities with a semaphore?