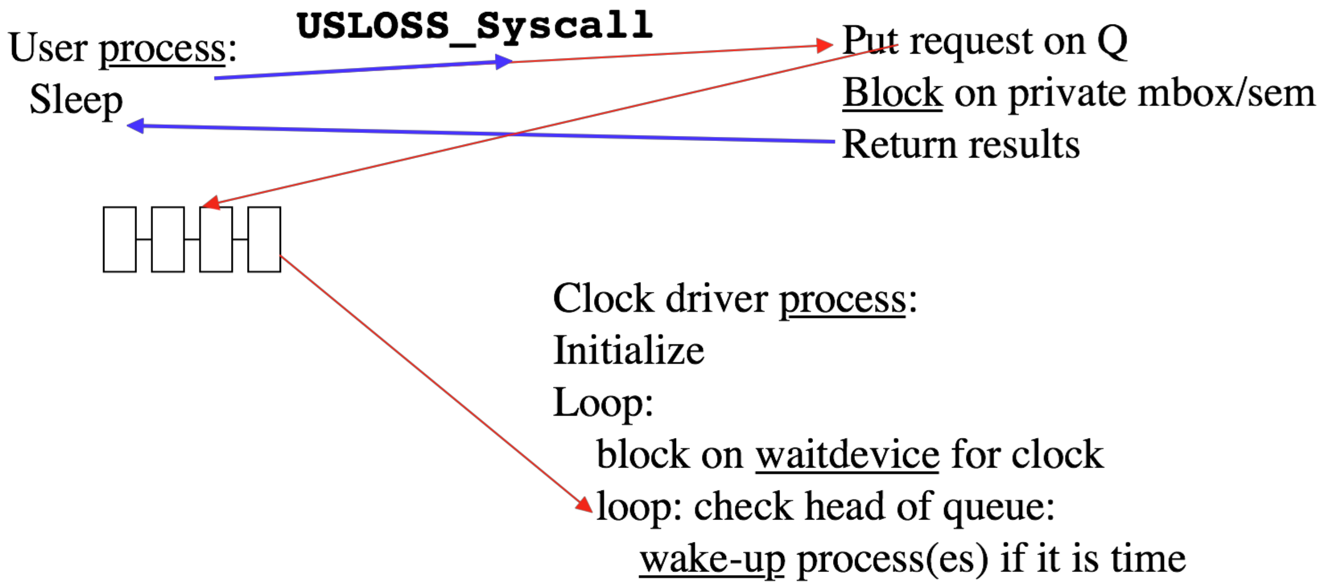
Phase 4 Notes and Hints

- Getting started:
  - Provided starter files and test cases for Phase 4 are located in: *~cs452/fall15/phase4*.
  - Your phase 4 code will be compiled into a library named: *libphase4.a* in your directory.
  - To execute a test case, you link the **`.o`** file of the test case with the *libphase4.a* and provided phase 1-4 libraries.
    - Typing '**`make`**' will create the *libphase4.a* library.
    - Typing '**`make test00`**', for example, will create an executable test case named *test00*.
    - You can use your own *libphase[1-3].a* libraries or mine during development.
    - Your phase 4 will be graded using my *libphase[1-3].a* libraries.

- Header files for Phase 4:
  - */home/cs452/fall15/include/phase4.h*: contains function prototypes and constants to be used in this phase.
  - */home/cs452/fall15/include/phase[1-3].h*: from previous phases.
  - */home/cs452/fall15/include/usloss.h*: contains function prototypes for USLOSS library functions, many useful constants.
  - */home/cs452/fall15/include/usyscall.h*: system call definitions, <u>expanded to include new ones from phase 4</u>.
  - *./usloss/include/libuser.h*: prototype definitions for the **`usyscall`** wrapper functions, <u>expanded for phase 4</u>.
  - Your data structures and constants for phase 4 will go into a local **`.h`** file that you will need to provide.

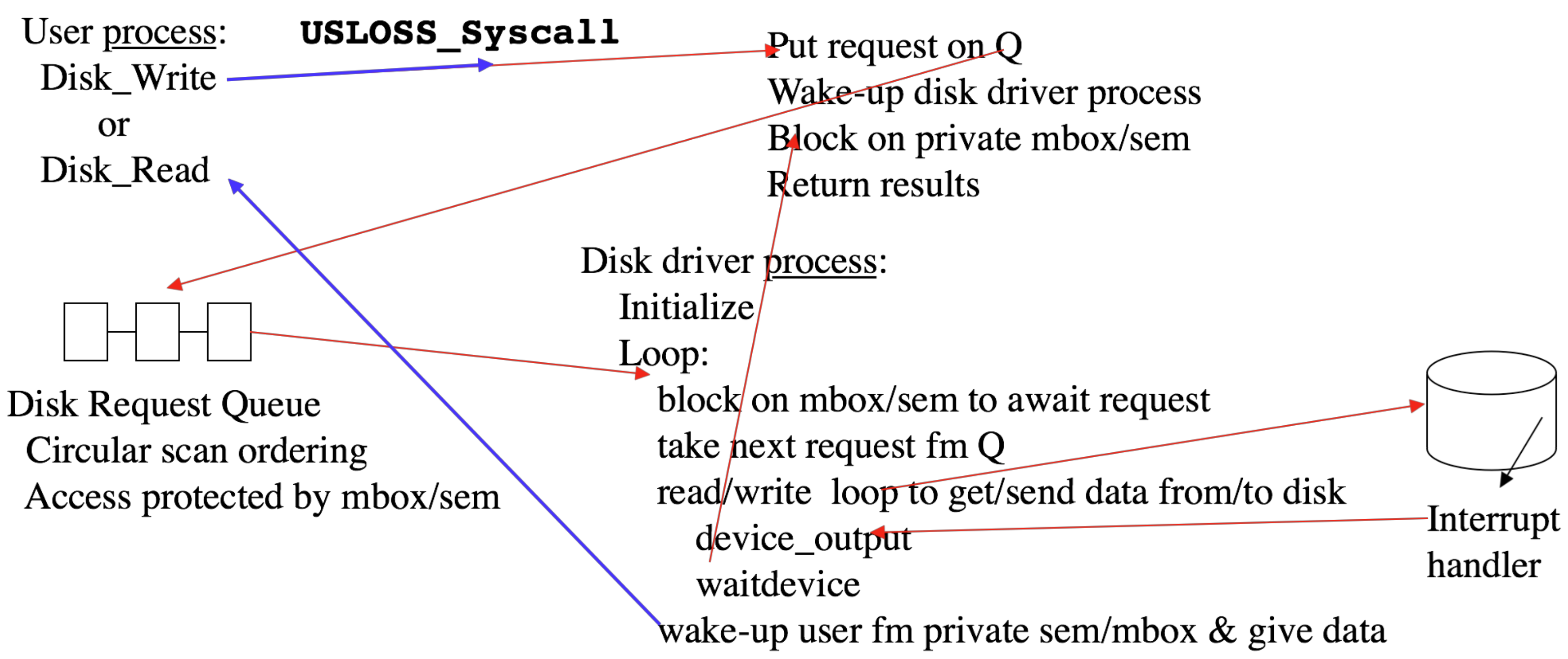
- **`start3()`** function:
  - The phase 3 library (yours or mine) will use **`fork1`** (not **`Spawn`**) to create a process at priority 1 that will execute the **`start3()`** code that you provide in phase 4. Thus, **`start3()`** is the entry point for phase 4. When **`start3()`** is called, there will be four processes already created: **`sentinel`**, **`start1`**, **`start2`**, and **`start3`**.
  - Initialize your phase 4 data structures, in particular, the phase 4 process table:
    - You will need a process table for phase 4. You cannot modify/extend the phase 1, 2, or 3 process tables! Use **`MAXPROC`** for the size of the phase 4 process table.
      - Note: When using my phase 1 library, you can use **`getpid() % MAXPROC`** to determine which slot in your phase 4 process table to use.
  - Initialize the appropriate elements of the **`systemCallVec`** array to point to the <u>new system call functions</u> that you are adding in phase 4.
  - Start, using **`fork1`**, the I/O driver processes, running at priority 2:
    - Clock driver: for sleeping processes.
    - Disk drivers: to handle read/write to/from the two disk units.
    - Terminal drivers: to handle read/write to/from the four terminal units.
  - Use **`spawnReal`** to start the test process: **`start4()`**. Note that **`start4`** will be the first user-mode process.
  - **`waitReal`** for **`start4()`** to finish.
  - Get rid of the various drivers; **`join`** with each.

- Clock Driver:
  - Sleep function, process requests delay for a specified number of seconds.
  - Process puts itself on a queue.
  - Clock driver checks on each clock interrupt to see which process(es) to wake up.



```
Clock driver process:
Initialize
Loop:
    block on waitdevice for clock
    loop: check head of queue:
        wake-up process(es) if it is time
```

- Disk Drivers:
  - Two disk devices in USLOSS. Need a driver for each.
  - User process makes requests via **`DiskWrite`**, **`DiskRead`**, or **`DiskSize`**.
  - Read/Write requests need to be optimized for seek (use Circular Scan).



- Terminals:
  - User process interface:
    - TermRead system call:
      - Receive a <u>line</u> of input from a mailbox associated with the indicated terminal.
    - TermWrite system call:
      - Send a <u>line</u> of output to the indicated terminal: use a mailbox to put the line into, or a semaphore-protected data structure.
      - Wait (on private semaphore or private mailbox) for the line to be written.
  - Each terminal handles both input and output, which can occur simultaneously; that is, on the <u>same</u> interrupt.
  - Input: collect individual characters into lines, give a line to user process, buffer up to 10 lines.
  - Output: user process requests output of a string; must send all the characters, one at a time, to the terminal.
  - **`start3()`**:
    - **`fork1`** a terminal driver (TermDriver) for each of the four terminals.
    - **`fork1`** a terminal reader (TermReader) for each of the four terminals.
    - **`fork1`** a terminal writer (TermWriter) for each of the four terminals.

- TermDriver (four instances):
  - Calls **`waitdevice()`** for that terminal.
  - Depending on result of **`waitdevice()`**:
    - Important note here: interrupt signals
      - <u>receipt</u> of a character, or
      - <u>completion of the send</u> of a character, or
      - <u>both</u>!!
    - Send received character to the character-in mbox (can also use a semaphore protected structure).
    - Send result of output of character to character-out mbox (can also use a semaphore protected structure).

- TermReader (four instances):
  - Collects individual characters from the character-in mbox (can also use a semaphore protected structure).
  - Builds “lines”, delimited by newlines, or when MAXLINE characters have been read.
  - Sends completed lines to a mailbox.
  - Buffers up to 10 lines. Begin discarding lines (not characters) when 10 limit is reached.

- TermWriter (four instances):
  - Receive a line of output from mbox or semaphore protected structure (was put there by user process doing the appropriate syscall)
  - Set terminal to have transmit interrupts enabled (see USLOSS manual, best not to have transmit interrupts enabled except when actually sending characters to the terminal).
  - On each interrupt that indicates **`DEV_READY`**, send one character.
  - When done with string:
    - Disable transmit interrupts.
    - Send result to user process via private mbox or a structure protected by a private semaphore.