Overview of s_osk

- **s_osk** simulates a simple OS running on a single CPU
- Simulated system calls (called by user processes)
  - `s_begin`, `s_end` User task startup/termination
  - `s_spawn`, `s_wait` Task creation/wait
  - `s_yield` Voluntary context switch
  - `s_getuptime` Get system clock value (virtual time)
  - `s_gettid`, `s_getptid` Return my task id/parent's task id
- Unix mechanisms used in implementation
  - Process creation, termination, synchronization
  - Inter-process communication (FIFOs)
  - Signals and signal handlers

Conceptual Model

- A **task** is a simulated process
  - One system task (s_osk)
  - Zero or more user tasks
- System task (s_osk)
  - Controls creation of all user tasks
  - Tracks task CPU usage
- The system is **non-preemptive**
  - A task retains control until it makes a system call
  - One Exception: A **timeout alarm** prevents runaway task
  - s_osk terminates when the timeout alarm has expired or when all user tasks have terminated
- There should be at most one active thread of control among the simulated tasks and s_osk

Example 1 (1)

- **user1.c** User Process
  - Initialize by calling `s_begin()`
  - Terminate by calling `s_end()`
  - `s_begin()` and `s_end()` are defined in `s_runlib.c`
  - Compile and link with user1.o

  ```c
  #include <s_osk.h>
  #include <stdlib.h>
  int main (void) {
    s_begin(); ... Compute ... s_end();
  }
  ```

- **s_osk** Simulated OS
  - Start execution: `s_osk -v 5 user1`
  - Set verbose level to 5
  - Fork/exec initial user task 'user1'
  - Initial task may create other tasks through `s_spawn()`
  - After fork/exec, wait for user1 to initialize and return control to s_osk
**Example 1 (2)**

**Control Structure**
- Control passes back and forth between `s_osk` and `user1` processes through messages

```
user1
s_begin() { write msg; read msg; }
  ... Compute ...
  s_end() { write msg; }
```

```
s_osk
Initialize;
Update procTbl;
fork/exec user1;
read msg; Update procTbl;
write msg;
read msg; Update procTbl;
... 
```

**Initialization Example**

```
user1: pid = 385
  tid = 1

Created by `s_osk`

"syscallFifo"

Created by `user1`

```
\(s\text{\_begin}(\) { write msg; \\
\} Create replyFifoNNNNNN; \\
\) Open "syscallFifo" RO;
\) Open "syscallFifo" RO;
\) Read REPLY msg from reply FIFO;
```

**Deadlock Startup**

```
s\_osk
Create "syscallFifo";
Open "syscallFifo" RO;
Fork/exec user1;
```

```
```

**SysCall Message Formats**

**Message Header**

```
struct callHdr {
  u_int16_t bodySz; // #bytes in msg body
  tid_t tid; // task id; pid if from s_begin
  u_int16_t sysCall; // system call number
};
```

- **System calls with 0-length bodies**
  » `s_begin`, `s_getptid`, `s_getuptime`

- **`s_end` and `s_yield` msg bodies**
  » struct timeval `vtime`; // accumulated virtual time

- **open()** is defined as a RENDEZVOUS
SysCall Reply Message Formats

- **s_begin**: Task id
- **s_getptid**: Task id
- **s_getuptime**
  - `struct timeval sysclock; // sum of all virtual times`
- **s_end** and **s_spawn**
  - `return code`

Example 2

**Control Structure**

```
user2
```

- **s_begin**
  - `(write msg; read msg;)`
  - `Update procTbl; fork/exec user1;`  
  - `write msg to user2; read msg; Update procTbl;`
  - ... Compute ...
- **s_yield**
  - `(write msg; read msg;)`
- **s_end**
  - `(write msg;)`
- **s_osk**
  - `Initialize; Read msg; Update procTbl;`
  - `write msg to user2; read msg; Update procTbl;`
  - ... user1 is at head of run queue
  - `write msg to user2; read msg;`

Example 3

```
user3
```

- **s_begin**
  - `(write msg; read msg;)`
  - `tid = s_spawn(...)`
  - `(write msg; read msg;)`
  - `... Compute ...`
- **s_yield**
  - `(write msg; read msg;)`
  - `... Compute ...`
- **s_end**
  - `(write msg;)`

**s_osk**

- `Initialize: Read msg; Update procTbl; fork/exec user1;`  
  - `write msg to user2; read msg; Update procTbl;`
  - ... user1 is at head of run queue
  - `write msg to user2; read msg;`
- After task T returns control etc.

Example 4

```
initialization; Update procTbl; fork/exec user2;  
```

- **s_begin**
  - `(write msg; read msg;)`
  - `Write msg to user2; Read msg; Update procTbl; fork/exec new task T;`  
  - `Read msg from T; // s_begin write msg to user2; read msg;`  
  - `... Task T at head of run queue etc.

Example 5

```
s_osk -v 5 user5 50000 60000
```

- **user5.c**
  - `s_begin();`
  - `s_spawn("./user5", "user5", argv[1], (char *)0);`  
  - `s_spawn("./user5", "user5", argv[2], (char *)0);`
  - `sleep(20); // for testing`
- **user5a.c**
  - `int calcPrime(int n) { ... Some code ... }`
  - `int main(int argc, char *argv[]) {`
    - `int n = atoi(argv[1]);`
    - `s_begin();`
    - `calcPrime(n);`
    - `s_end();`
    }