EGOS and process.c

CS 4411

Spring 2020

Announcements

- EGOS source code update
- Cornell Undergraduate Research advertisements


CORNELL UNDERGRADUATE
RESEARCH BOARD PRESENTS
Grad School Demystified
Mix and mingle with a panel of current graduate students to learn more about applying and going to graduate school!
MARCH 1ST
3 PM
KLARMAN HALL
KG70
@curb_cornell
curb.cornell.edu
facebook.com/cornellcurb
Questions? Contact jg999@cornell.edu
Presented by
CU
RB

Cornell Undergraduate Research Board invites you to apply to present at

Spring Symposium

WHAT?

Cornell's largest undergraduate research event

WHEN?

Wednesday, April 15, 2020
5-7 PM

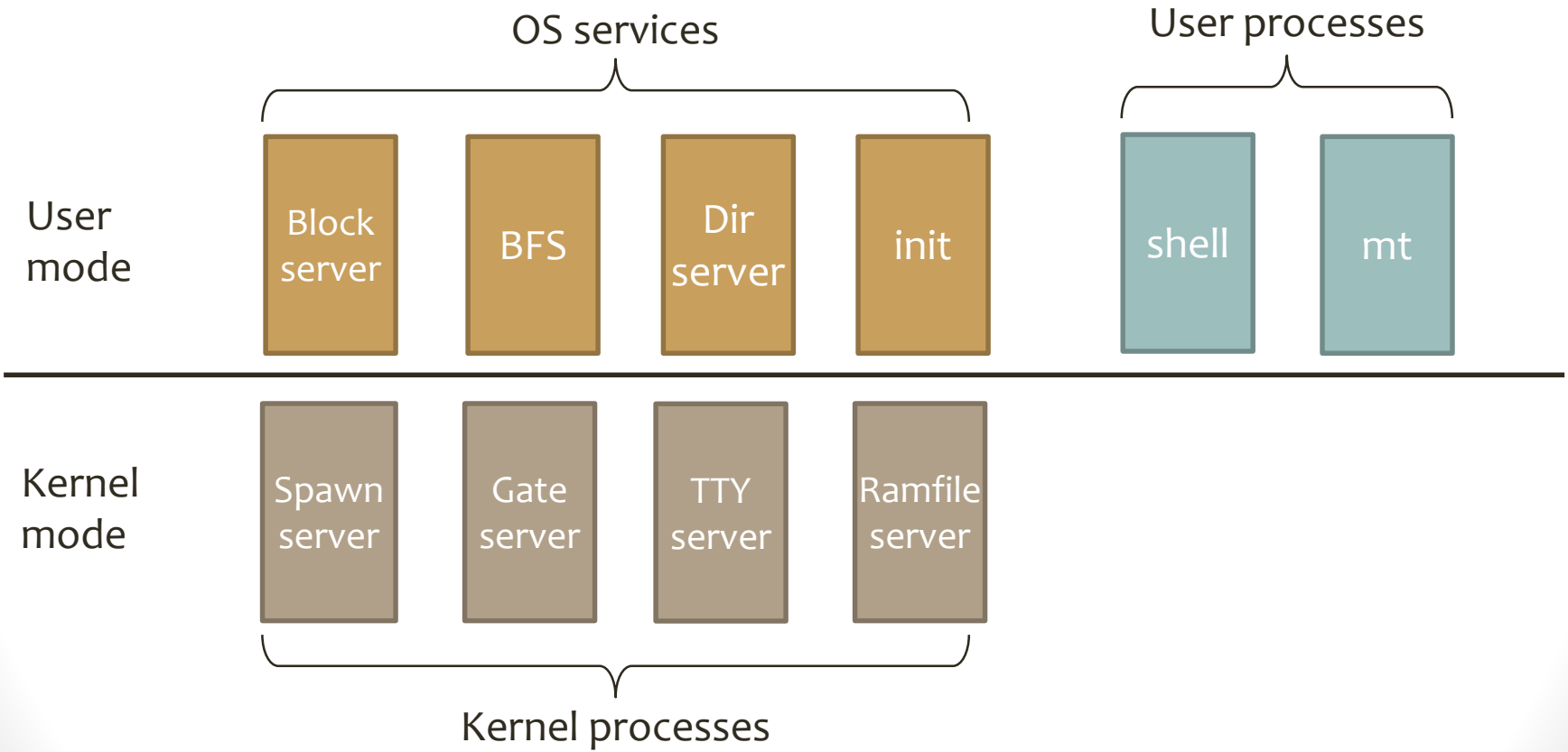
HOW?

Submit your research by Friday, March 20, 2020 11:59:59 PM EST to TINYURL.COM/CURB2020

Outline for Today

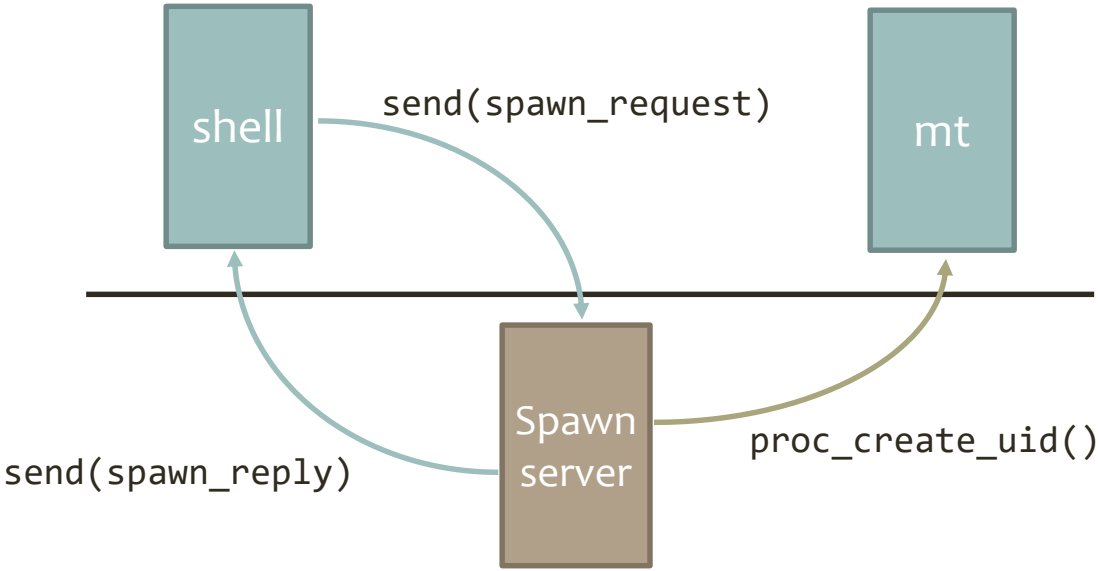
- EGOS Concepts
  - Kernel and user processes
  - Message passing
- Process.c overview

EGOS: A Microkernel



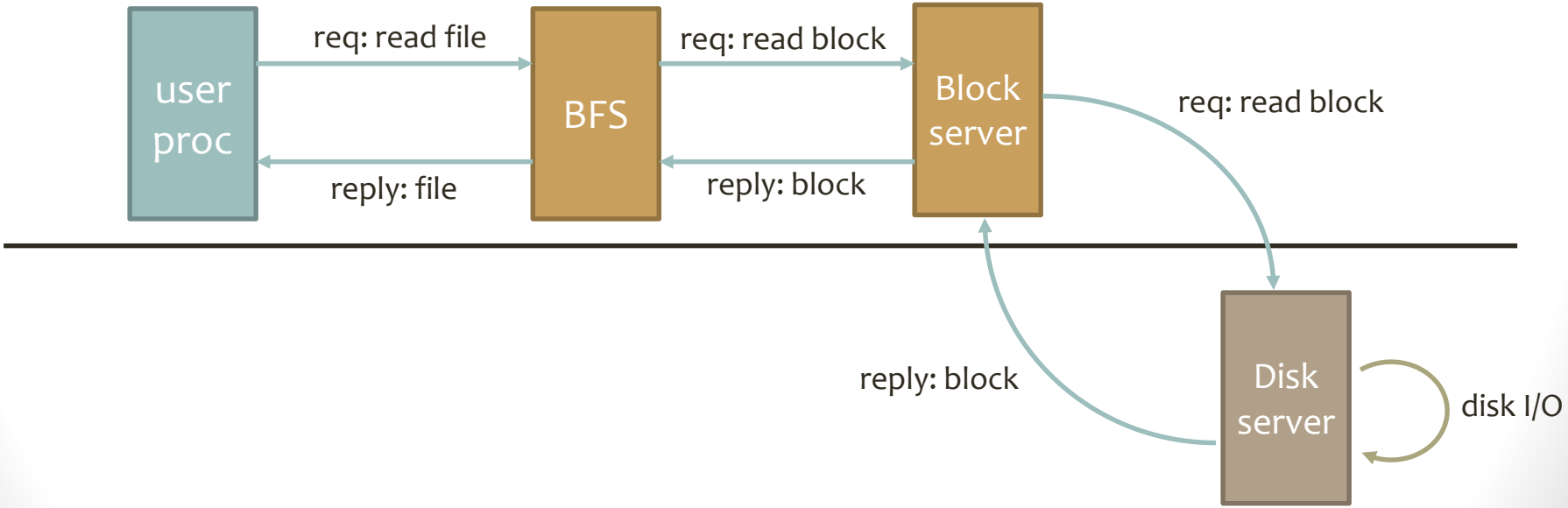
Message Passing

- Processes communicate by sending messages
- Most system calls are actually message request/reply pairs



I/O with Messages

- Reading a file: standard example of “waiting for I/O”



Message System Calls

```
int sys_send(gpid_t pid, enum msg_type mtype,
    const void *msg, unsigned int size);
```

Send a message to process ID `pid`, with contents in buffer `*msg` of size `size`. Message type `mtype` is either REQUEST or REPLY.

```
int sys_recv(enum msg_type mtype, unsigned int
    max_time, void *msg, unsigned int size, gpid_t
    *src, unsigned int *uid);
```

Block and wait for a message of type `mtype` for at most `max_time` ms. The message will be placed in `*msg`, the sender's process ID and user ID will be placed in `*src` and `*uid`

Message System Calls

```
int sys_rpc(gpid_t pid, const void *request,
    unsigned int reqsize, void *reply,
    unsigned int repsize);
```

Send a message to process `pid` and immediately block until a reply is received. The reply will be placed in `*reply`.

Outline for Today

- EGOS Concepts
  - Kernel and user processes
  - Message passing
- **Process.c overview**

Public Interface of process.c

```
gpid_t proc_create_uid(gpid_t owner, char *descr,
    void (*fun)(void *), void *arg, unsigned int uid);
```

Creates a new process with parent `owner`, which will run function `fun` with argument `arg`. User ID 0 indicates root.

```
void proc_kill(gpid_t killer, gpid_t pid, int status);
```

Kills process `pid`, giving it exit status `status`, provided `killer` is allowed to kill that process.

```
void proc_dump();
```

Prints out status of all running processes – the ctrl-L command

Message Passing Functions

```
bool proc_recv(enum msg_type mtype, unsigned int
    max_time, void *contents, unsigned int *psize,
    gpid_t *psrc, unsigned int *puid);
```

Implements sys_recv(). Waits for a message to be delivered to one of this process's "mailboxes"

```
bool proc_send(gpid_t src_pid, unsigned int
    src_uid, gpid_t dst_pid, enum msg_type mtype,
    const void *contents, unsigned int size);
```

Implements sys_send(). Can be called by the kernel in an interrupt handler, so not necessarily a send from the current process

Process.c Memory Management

- Design decision: Don't spend time allocating/freeing PCBs during normal execution
- All PCBs statically allocated at boot time:
  `static struct process proc_set[MAX_PROCS];`
- PCBs marked as "free" with `state = PROC_FREE;`
- On `proc_alloc()`, grab a free PCB from the free list, zero it out
- On `proc_release()`, mark PCB as free and return it to free list

Let’s Look at Some Code