IPC IV: epoll

CS 241

Nov. 4, 2013

I/O Multiplexing

- By default: **read()** / **fread()** are blocking calls.
  - …if no data is available, the process will be moved to the BLOCKED state until data is available.

- In order to read() from multiple files in one thread at one time, **I/O multiplexing** is required.
  - **epoll()**: *monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready“.*

epoll() Overview

- Usage of epoll():
  - *Create an epoll instance via **epoll_create()***
  - *Register each file descriptor to watch via **epoll_ctl()***
  - *Use **epoll_wait()** to block until an fd is ready*
  - *(Replaces both select() and poll() system calls.)*

epoll() Overview

- **epoll_ctl():**

```
int epoll_ctl(int epfd, int op, int fd,
              struct epoll_event *event);

op: EPOLL_CTL_ADD: Add to the epoll set
    EPOLL_CTL_MOD: Modify the epoll set
    EPOLL_CTL_DEL: Delete from the epoll set

event:
  struct epoll_event {
     uint32_t     events;      /* Epoll events */
     epoll_data_t data;        /* User data */
  };

  typedef union epoll_data {
     int fd;
     ...   // ...other stuff we will not use.
  } epoll_data_t;
```

epoll() Example

	Process 1	***Process 2***
0s:	*A*	
1s:		*B*
2s:		*C*
3s:	*D*	

epoll() Example

```
void one(int write_fd)
{
    sleep(1);
    write(write_fd, "B", 1);
    sleep(1);
    write(write_fd, "C", 1);
    close(write_fd);
}

void two(int write_fd)
{
    write(write_fd, "A", 1);
    sleep(3);
    write(write_fd, "D", 1);

    close(write_fd);
}
```

```
void main() {













}
```