Embedded Operating Systems

Agenda

- Signals
- Message Queue

Signals

Signal related Systems calls

sigaction() syscall

- sigaction() is extended version of signal() in Linux.
- ret = sigaction(signum, &new_sigaction, &old_sigaction);
  - arg1: signal number of the signal to be handled.
  - arg2: (in param) address of sigaction struct that hold address of signal handler function.
  - arg3: (out param) address of sigaction struct to collect address of old signal handler. If NULL, then old sigaction is not returned.
  - returns: 0 on success.
- man sigaction
  - struct sigaction
    - sa_handler -- sig handler fn -- void handler(int);
    - sa_sigaction -- sig handler fn -- void handler(int,siginfo_t*,void*);
    - sa_flags -- SA_SIGINFO to indicate that sa_sigaction handler given.
    - sa_mask
    - sa_restorer (deprecated)
- To handle signal (method1)
  - step 1: implement signal handler function

```
void sigint_handler(int sig) {
    // ...
}
```

- step 2: register signal handler function

```
struct sigaction sa;
memset(&sa, 0, sizeof(struct sigaction));
sa.sa_handler = sigint_handler;
ret = sigaction(SIGINT, &sa, NULL);
```

- To handle signal (method2)
  - step 1: implement signal handler function

```
void sigint_handler(int sig, siginfo_t *si, void *param) {
    // ... si contains info about the signal e.g. si_pid (signal sender pid)
}
```

  - step 2: register signal handler function

```
struct sigaction sa;
memset(&sa, 0, sizeof(struct sigaction));
sa.sa_flags = SA_SIGINFO;
sa.sa_sigaction = sigint_handler;
ret = sigaction(SIGINT, &sa, NULL);
```

Signal delivery internals

- Refer slides.
- Refer Design of UNIX Operating System (Bach).

Signal masking

sigset_t

- sigset_t represent signal mask field.
- Logically it is 64-bit field -- 1 bit for each signal.
- sigemptyset(&set); --> make all signal bits 0
- sigfillset(&set); --> make all signal bits 1
- sigaddset(&set, signum); --> make signal bit 1 corresponding to given signal number
- sigdelset(&set, signum); --> make signal bit 0 corresponding to given signal number

sigprocmask() syscall

- Change the signal mask field to mask/unmask signals of the current process (till next call to the sigprocmask()).
- sigprocmask(mode, &new_sigset, &old_sigset)
    - arg1: mode=SIG_SETMASK to set new signal mask in the PCB of current process.
    - arg2: new signal mask. Signals to be masked should be 1 and other signals should be 0.
    - arg3: old signal mask (out param).
    - returns: 0 on success.

sigsuspend() syscall

- Change the signal mask field to mask given signals and pause the execution of the process until any of the unmasked signal is received.
- Once unmasked signal is received, its handler (user-defined or default) will be executed and old signal mask will be restored. The process will continue further (if not terminated due to signal handler).
- sigsuspend(&set);
    - arg1: temporary signal mask field for which process will be suspended.

pause() syscall

- Wait for any signal whose handler to be executed or process to be terminated.

EINTR

- If a syscall blocks a process in INTERRUPTIBLE_SLEEP (e.g. read()) and a signal arrives, which can terminate the process or whose signal handler to be executed, then process is forcibly woken up. That syscall returns with error EINTR i.e. Interrupted System Call.

Message Queue

- Used to transfer packets of data from one process to another.
- It is bi-directional IPC mechanism.
- Internally OS maintains list of messages called as "message queue" or "mailbox".
- The info about msg que is stored in a object. It contains unique KEY, permissions, message list, number of messages in list, processes waiting for a message to receive (waiting queue).
- Refer slides and design of UNIX Operating System (Bach).

Message Queue Syscalls

msgget()

- Create message queue object.
- mqid = msgget(mq_key, flags);
  - arg1: unique key
  - arg2: IPC_CREAT | 0600 to create new message queue.
  - returns message queue id on success

msgctl()

- Get info about message queue or destroy message queue
- msgctl(mqid, IPC_RMID, NULL) -- to destroy message queue
  - arg1: message queue id
  - arg2: commad = IPC_RMID to destroy message queue
  - arg3: NULL (not required while destroying message queue)

msgsnd() - send message into que

- Send message in the message queue.
- ret = msgsnd(mqid, msg_addr, msg_size, flags)
  - arg1: message queue id
  - arg2: base address of message object
  - arg3: message body size
  - arg4: flags (=0 for default behaviour)
  - returns 0 on success.

msgrcv() - receive message from que

- Receive message from the message queue of given type.
- ret = msgrcv(mqid, msg_addr, msg_size, msg_type, flags)
  - arg1: message queue id
  - arg2: base address of message object to collect message (out param)
  - arg3: message body size
  - arg4: type of message to be received
  - arg5: flags (=0 for default behaviour)
  - returns number of bytes (body) received on success.

IPC commands

- ipcs -- to show IPC (Shared memory, Semaphore, Message queue) status
- ipcrm -- to delete IPC (Shared memory, Semaphore, Message queue) object

Assignments

1. The child process send two numbers to the parent process via message queue. The parent process calculate the sum and return via same message queue. The child process print the result and exit. The parent process wait for completion of the child and then exit.