Announcement

• PA2 interview scheduler will be opened by the end of TODAY !!!
• Grading policy: 40% coding + 60% interview
• What to prepare for the interview?
  • Laptop
  • Submitted source code
  • Knowledge … for example
    • Test cases
    • Used commands (e.g., mknod)

• No office hours next week
Week 5: Inter-Process Communication
Process Cooperation

Reasons for providing an environment that allows process cooperation:

- Information Sharing
- Computation Speedup
- Modularity
- Convenience

Cooperating processes needs **IPC** mechanism !!!
IPC Mechanisms

• Signals / Interrupts

• Message Passing
  • Pipes
  • Sockets

• Remote Procedure Calls

• Shared Memory
IPC Mechanisms

• Signals / Interrupts

• Message Passing
  • Pipes
  • Sockets

• Remote Procedure Calls

• Shared Memory
Message Passing

• Communication takes place by means of messages exchanged between the cooperating processes.

• Advantages:
  • Useful for exchanging small amounts of data, because no conflicts need be avoided
  • Easy to implement in a distributed system

• Disadvantages:
  • Time-consuming as typically implemented using system calls
Message Passing (Linux pipes)

• A pipe is a **unidirectional** (in memory) I/O channel between two processes.

• Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.

• Read and write system calls are used for I/O through pipes.
Message Passing (Linux pipes)

• A pipe is created using the `pipe()` system call:

```
int pipe(int fildes[2]);
```

• Library: `unistd.h`

• `fildes`: stores two file descriptors returned, which refer to the two ends of the pipe.
  • `fildes [0]` refers to the read end of the pipe.
  • `fildes [1]` refers to the write end of the pipe.
Pipe Example

Main Process

Messages are streams of numeric data

Pipe 1

mcpipe1[0] mcpipe1[1]

Pipe 2

mcpipe2[0] mcpipe2[1]

"Cube" file

Child Process 2

"Square" file

Child Process 1

mcpipe2[0] mcpipe1[0]

mcpipe1[1] mcpipe2[1]
Shared Memory

• A region of memory is established and shared by cooperating processes.
• Processes can then exchange information by reading and writing data to the shared region.

• Advantages:
  • Fast as system calls are required only to establish shared-memory
  • Useful for exchanging large amounts of data

• Disadvantages:
  • Complicated to implement in a distributed system
Shared Memory

• Create a shared memory segment or get the ID of an existing shared segment

\[
\text{int shmget(key_t key, size_t size, int shmflg);}\]

• Other processes (if permitted) can access.

• Library: \textit{sys/ipc.h} and \textit{sys/shm.h}
  
  • \textit{key}: a value used to share between processes
  
  • \textit{size}: size of the shared memory segment
  
  • \textit{shmflg}: options for segment creation
Shared Memory

• Attach the shared segment to a process address space

```c
void *shmat(int shmid, const void *shmaddr,… int shmflg);
```

• Once attached, a process can read or write to the segment.

• Library: `sys/types.h` and `sys/shm.h`
  • `shmid`: id of the shared memory segment
  • `shmaddr`: address in the process space
  • `shmflg`: options for shared memory permission
Shared Memory

• Detach the shared memory segment located at the \texttt{shmaddr} address

\begin{verbatim}
int shmdt(const void *shmaddr);
\end{verbatim}

• Library: \texttt{sys/types.h} and \texttt{sys/shm.h}
  • \texttt{shmaddr}: attaching address in the process space
Shared Memory Example

Process
SERVER

Code

Data

Heap

Stack

Write

shmaddr

Key = 5678

Shared Memory

Process
CLIENT

Code

Data

Heap

Stack

Key = 5678

shmaddr

Read

CSCI 3753 Spring 2020
Week 5 – Checklist

☐ Discuss IPC
☐ Read more about IPC