Advanced Unix Programming

Module 06

Raju Alluri
askraju @ spurthi.com

Advanced Unix Programming: Module 6

- Data Management
  - Basic Memory Management
  - File Locking

Basic Memory Management

- Memory allocation using malloc()
  - Allocate a specific number of bytes

```
void *malloc(size_t size);
```

  - Allocate memory for size number of bytes
  - The contents of the memory allocated are indeterminate
    - So the calling program should take care of initializing the memory

Basic Memory Management

- Memory allocation using calloc()
  - Allocate memory for a specific number of objects, each of a specific size

```
void *calloc(size_t nobj, size_t size);
```

  - Allocate memory for `nobj` number of objects, each of which is of size `size` number of bytes
  - The allocated space is reinitialized to bits of value 0

Basic Memory Management

- Memory allocation using realloc()
  - Reallocate given memory by either increasing or decreasing the size

```
void *realloc(void *ptr, size_t newsize);
```

  - Increase/Decrease the memory pointed to by ptr to newsize.
  - If the size is increased, it may involve moving all the current contents pointed to by ptr to a new location
  - If the size is increased, the contents of the augmented memory is indeterminate

Basic Memory Management

- Freeing the memory

```
void free(void *ptr);
```

  - Free the memory pointed to by ptr.
  - The memory might be reused for subsequent allocation operations

File and Record Locking

- Locking
  - Used to determine the control of a process or thread over a block of memory, a file or part of a file
  - Owner of the lock can perform the operation and release a lock
  - Other processes and threads can wait until a lock is released
  - File Locking
    - Process of locking an entire file
  - Record Locking
    - Process of locking part of a file, from a start location to end location

File and Record Locking

- Case Study
  - Assigning sequence numbers
    - The current number is stored in a file
    - Process interested in a sequence number
      - Read the number from the file and keep it for using
      - Increment the number and write it back to the file
    - Q: What if two processes read the number from the file simultaneously

File and Record Locking

- Advisory Locking
  - OS remembers the lock on a file
  - Doesn't enforce writing on to locked files
  - Useful for cooperating processes
- Mandatory Locking
  - Operating system enforces the locks
  - Enabling a file for mandatory locking
    - Set Group exectue bit to be off
    - Set set-group-ID bit to on

Record Locking using fcntl()

- Lock specific part of the file

```
int fcntl(int fd, int cmd, ... /*struct
  flock *arg */);
```

  - The flock structure is defined by

```
struct flock {
  short l_type; /* F_RDLCK, F_WRLCK, F_UNLCK*/
  short l_whence;
  off_t l_start;
  off_t l_end;
  pid_t l_pid;
}
```

  -

Record Locking using fcntl()

- Lock specific part of the file

```
int fcntl(int fd, int cmd, ... /*struct
  flock *arg */);
```

  - Cmd is one of
    - F_SETLK: Set a read-write lock (non-waiting)
    - F_SETLKW: Set a read-write lock (waiting)
    - F_GETLK: Get attributes of the existing lock

Other Locking Techniques

- Other Unix Locking techniques
  - Use a specific file as a lock to contents of another file
    - Use the link() system call to create a lock file
      - We can use a temp filename (which contains the pid) to create a temp file and then create a link to it
      - Fails if the link/file already exists
  - Create a lock file using creat() with no write access
    - This will prevent other processes (even with super user permissions) to create the same file
  - Create the lock file using open() and give the flags `O_CREAT` and `O_EXCL`.