LSN 13 – Creating Processes

• `fork()` system call
  – Returns PID of the child process created
  – The new process is created with its own copies of the parent’s program text, data, and stack segments

```
childPID = fork();
if( childPID == 0)
{
    //execute code for the child here
    exit(0);
}
//execute code for all processes here
```
LSN 13 – Creating Processes

• Child and parent processes operate concurrently in their own separate address spaces

• `wait()` / `waitpid()`
  – Enables the parent process to detect when one of its child processes terminates
  – Details of the terminating child process’s status may be returned to the parent via a value parameter passed to `wait()`, or it may be ignored by the parent
  – When the parent executes the `wait()` call, it acknowledges the child process termination, allowing the OS to release the relevant data structures
LSN 13 – POSIX Thread Package

- **pthread.h**
  - `pthread_t` type
  - `pthread_create()`
    - Create a new thread to run in the address space of the calling thread
    ```c
    int pthread_create( pthread_t *thread,
                       const pthread_attr_t *attr,
                       void *(*start_routine)(void),
                       void *arg );
    ```
    - Each thread is represented by an identifier, of type `pthread_t`
    - Returns 0 on success
LSN 13 – POSIX Thread Package

- **pthread_join()**
  - Parent thread can wait for a child thread to terminate for synchronization
  - It indicates that the caller wishes to block until the thread being joined exits
  - Required to deallocate thread resources after it terminates

```c
int pthread_join( pthread_t thread,
                 void **status );
```

- **pthread_exit()**
  - Called by child thread to terminate
  - Status information can be transmitted to parent thread through return value

```c
void *pthread_exit( void *);
```
LSN 13 – POSIX Thread Package

• Example:

```c
int main()
{
    pthread_t my_thread;
    int *ret_val;
    char *a_list;
    . . .

    if( !pthread_create( &my_thread, NULL,
                        worker_thread, (void *)a_list) )
    {
        printf(stderr, . . . );
    }
    . . .
    //wait for the child to terminate
    if( !pthread_join( my_thread, &ret_val ) )
    {
        // error return
    }
}
```
LSN 13 – POSIX Thread Package

```c
void *worker_thread( void *arg)
{
    int *ret_val;

    ...

    //work is completed, terminate
    pthread_exit( ret_val);
}
```
LSN 13 – POSIX Thread Package

• Synchronization primitives within pthread library
  – Mutex
  – Condition variables
  – Read/write lock

• `pthread_mutex_t` type
  – Created as an un-owned resource, which threads can obtain and relinquish ownership of
LSN 13 – Mutex Operation

- Mutex creation

```c
pthread_mutex_init( pthread_mutex_t *mutex,
                    const pthread_mutex_attr_t *attr );
```

- Mutex release

```c
pthread_mutex_destroy( pthread_mutex_t *mutex );
```

- Mutex synchronization

```c
pthread_mutex_lock( pthread_mutex_t *mutex );
pthread_mutex_unlock( pthread_mutex_t *mutex );
```
LSN 13 – IPC and Pipes

• A pipe is represented in the kernel by a file descriptor
  – Process creates a pipe from a 2 element integer array
    
    int pipeID[2];
    ...
    pipe( pipeID );
    ...
  – Kernel creates pipe as a kernel FIFO data structure with two file identifiers
    • pipeID[0] is a file pointer to the read end of the pipe
    • pipeID[1] is a file pointer to the write end of the pipe
For multiple processes to use an anonymous pipe, a common ancestor must create the pipe. Multiple processes created using the **fork** command will be provided a copy of the open file table. Information is transferred to/from a pipe using asynchronous commands on associated pipe ends:

- `read()`
- `write()`

Pipe ends can be closed per process specifications.
Example:

```c
pipe(pipeID);

if( fork() == 0) // the child process
{
    . . .
    close( pipeID[1] ); // child process closes write
                        // side of pipe
    read(pipeID[0]), childbuf, len);
    /* process the message in childBuf */
    . . .
}
else // the parent process
{
    . . .
    close( pipeID[0] ); // parent process closes read
                        // side of pipe
    /* create a message to send to child process */
    write(pipeID[1], msgToChild, len);
    . . .
}
```
LSN 13 – Homework

• Reading
  – Chapter 6.8

• Assignment
  – HMWRK6.pdf
    • Must use the “g++” compiler
    • Include the “unistd.h” file

• References
  – www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html