Operating System Calls[*]

한양대학교 ERICA
소프트웨어학부 오희국[†]

2020년 3월 1일

Process

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
pid_t getpid(void);
pid_t getppid(void);

#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int sig);

#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *wstatus);
pid_t waitpid(pid_t pid, int *wstatus, int options);
int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);

#include <unistd.h>
extern char **environ;

int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char *const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);

#include <stdlib.h>

void exit(int status);
```

[*]이 자료는 한양대학교 ERICA 소프트웨어학부 학생이 과제를 수행하는데 도움을 주기 위해 만들었습니다.
[†]hkoh@hanyang.ac.kr

Shared Memory

```
#include <sys/mman.h>
#include <sys/stat.h>        /* For mode constants */
#include <fcntl.h>           /* For O_* constants */

int shm_open(const char *name, int oflag, mode_t mode);
int shm_unlink(const char *name);

#include <unistd.h>
#include <sys/types.h>

int truncate(const char *path, off_t length);
int ftruncate(int fd, off_t length);

#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
int munmap(void *addr, size_t length);
```

Pipe

```
#include <unistd.h>

int pipe(int pipefd[2]);

#define _GNU_SOURCE   /* See feature_test_macros(7) */
#include <fcntl.h>    /* Obtain O_* constant definitions */
#include <unistd.h>

int pipe2(int pipefd[2], int flags);

#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(const char *pathname, mode_t mode);
```

Pthread

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
int pthread_join(pthread_t thread, void **retval);
void pthread_exit(void *retval);
pthread_t pthread_self(void);
int pthread_cancel(pthread_t thread);
```

```
void pthread_testcancel(void);
int pthread_detach(pthread_t thread);

int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_getdetachstate(const pthread_attr_t *attr, int *detachstate);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
int pthread_attr_getstack(const pthread_attr_t *attr,
                          void **stackaddr, size_t *stacksize);
int pthread_attr_setstack(pthread_attr_t *attr,
                          void *stackaddr, size_t stacksize);
int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
                              size_t *restrict stacksize);
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
int pthread_attr_setschedparam(pthread_attr_t *attr,
                               const struct sched_param *param);
int pthread_attr_getschedparam(pthread_attr_t *attr,
                               struct sched_param *param);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
int pthread_attr_getschedpolicy(const pthread_attr_t *attr, int *policy);
int pthread_attr_setinheritsched(pthread_attr_t *attr,
                                 int inheritsched);
int pthread_attr_getinheritsched(const pthread_attr_t *attr,
                                 int *inheritsched);

#include <signal.h>

int pthread_kill(pthread_t thread, int sig);
```

Mutex Locks

```
#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
                       const pthread_mutexattr_t *restrict attr);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);

int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
                              int *restrict type);
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict attr,
                                  int *restrict protocol);
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
                                  int protocol);
int pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict attr,
                                 int *restrict pshared);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
                                 int pshared);
int pthread_mutexattr_getrobust(const pthread_mutexattr_t *attr,
                                int *robustness);
int pthread_mutexattr_setrobust(const pthread_mutexattr_t *attr,
                                int robustness);
int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *restrict attr,
                                     int *restrict prioceiling);
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
                                     int prioceiling);

#include <pthread.h>
#include <time.h>

int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
                            const struct timespec *restrict abs_timeout);
```

Semaphores

```
#include <fcntl.h>           /* For O_* constants */
#include <sys/stat.h>        /* For mode constants */
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag,
                mode_t mode, unsigned int value);
int sem_close(sem_t *sem);
int sem_unlink(const char *name);

int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_destroy(sem_t *sem);

int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout);
int sem_post(sem_t *sem);
int sem_getvalue(sem_t *sem, int *sval);
```

Condition Variables

```
#include <pthread.h>

int pthread_cond_destroy(pthread_cond_t *cond);
int pthread_cond_init(pthread_cond_t *restrict cond,
                      const pthread_condattr_t *restrict attr);
```

```
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int pthread_cond_wait(pthread_cond_t *restrict cond,
                      pthread_mutex_t *restrict mutex);
int pthread_cond_timedwait(pthread_cond_t *restrict cond,
                           pthread_mutex_t *restrict mutex,
                           const struct timespec *restrict abstime);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);

int pthread_condattr_destroy(pthread_condattr_t *attr);
int pthread_condattr_init(pthread_condattr_t *attr);
int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
                                int *restrict pshared);
int pthread_condattr_setpshared(pthread_condattr_t *attr,
                                int pshared);
int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
                              clockid_t *restrict clock_id);
int pthread_condattr_setclock(pthread_condattr_t *attr,
                              clockid_t clock_id);
```