C Multiprocessing & Multithreading Cheat Sheet

by Gorge97 via cheatography.com/154085/cs/33088/

Multiprocessing

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

int fork(void)

Fork the current process creating a child. Return 0 in the child process or child ID for the parent

int getpid(void)

Return the ID of the calling process

int getppid(void)

Return the ID of the parent of the calling process

void exit(status)

Process termination

unlisgned int sleep(unsigned int seconds)

Pause the execution of the calling process for *seconds* seconds or until a signal is received

Process Syncronization

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

pid_t wait(*-status)	Wait until a child process terminate the execution; On success return the pid of the child, on failure return -1; *status* is the address of the variable cointaining the exit status
pid_t waitpi-d(pid_t pid, int *status, int options)	Wait until the child specified with the *pid* argument terminate the execution; *status* is the exit status of the terminating process;

Shared Memory

#include <sys/shm.h>

int shmget (key_t key, int size, int shmflg)	Create a shared memory or connec to to an existing segment; *key* is a numeric key assigned to the segment. If IPC_PRIVATE is used the segment can be only used by parent and children; *size* is the size of the memory segment; *shmflg* is a flag field: IPC_CREATE create a new segment, IPC_EXCL cause the command to fail if the segment already exist. Return the shared memory segment id or -1 if fail.

Shared Memory (cont)

void *shmat(int shmid, const void* shmaddr, int shmflg)	Attach to the shared memory segment and return the address. *shmid* is the shared memory segment id; *shmadr* is the variable where the address of the segment is stored; *shmflg* is used to specify the access permissions for the shared memory segment and to request special attachment conditions, such as a read-only segment.
int shmdt(-const void *shmaddr)	Detaches the shared memory segment located at the address specified by shmaddr from the address space of the calling process.
int shmctl(int shmid, int cmd, struct shmid_ds *buf)	Performs the control operation specified by *cmd* on the shared memory segment whose identifier is given in shmid. Typical usage: shmctl-(shmid, IPC_RMID, 0); Remove shared memory segment

C Multiprocessing & Multithreading Cheat Sheet

by Gorge97 via cheatography.com/154085/cs/33088/

Message Queue

#include <sys/msg.h>

int msgget(key_t key, int flag)	Creates a message queue. *key* is an integer that specifies the queue key; *flag* indicates creation conditions and access permissions (same as shmget). Return a message queue identifier or -1 in case of failure.
int msgsnd(int msqid, const void *msgp, size_t nbytes, int flag);	Send a message on the queue. *msqid* is the identifier returned by msgget command; *msgp* is the pointer to the message struct. *Struct msgbuf {* *long mtype;* *char mtext [TEXT LENGHT];* *};* *nbytes* is the maximum lenght of the message;

Message Queue (cont)

int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtype, int msgflg)	Receive a message from the queue. *msqid* corresponds to the message queue identifier; *msgp* is the pointer to a message struct (the same used in msgsnd) ; *msgsz* is the number of bytes to read; *msgtype* is used to filterthe message in the queue. If != 0 read only a message with the same id on the queue; *msgflg* is a flag that modify the behavior of the command. Using IPC_NOWAIT return immediately of no message is found. The function return the number of bytes read or -1 if unsuccessful
int msgctl(int msqid, int cmd, struct msqid_ds *buf)	Modifies the properties of the queue or delete it. *msqid* corresponds to the message queue identifier. Typical usage: *msgctl(msgid, IPC_RMID, 0)*; Removes queue

Unnamed Pipes

#include <unistd.h>

int pipe(int fd[2])	Creates an unnamed pipe (unidirectional). *fd[2]* are two descriptor associated with the “read” end of the pipe (fd[0]) and with the “write” end of the pipe (fd[1]). Return 0 if the kernel could allocate enough space, -1 otherwise.
int write (int fd, char * buf, int size)	The classic write function is used to write inside the buffer. *fd* in this case is the descriptor of the write end of the pipe.
int read (int fd, char * buf, int size)	The classic read function is user to read data from the pipe. *fd* is the descriptor of the read end of the pipe.

Named Pipes or FIFO

#include <fcntl.h>

Named Pipes or FIFO (cont)

int mkfifo-(const char *path, mode_t mode);	Creates a named pipe (FIFO) *path* corresponds to the name of the pipe; *mode* corresponds to the permission mode flags.
int open(const char *path, int flags)	Open the fifo defined by the name saved in *path*; for the *flag* parameter use O_WRONLY for a write only pipe or O_EDONLY for a read only pipe.
int write (int fd, char * buf, int size)	The classic write function is used to write inside the buffer. *fd* in this case is the descriptor of the write end of the pipe.
int read (int fd, char * buf, int size)	The classic read function is user to read data from the pipe. *fd* is the descriptor of the read end of the pipe.

Signals

#include <signal.h>

Signals (cont)

int kill(pid_t pid, int sig)	Send a signal to the process indicated by the parameter pid. *pid* is the pid of the process that will receive the signal; if set to 0 all the processes in the process group of the current process receive the signal; *sig* is the type of the signal
sighan-dler_t signal(int signum, sighan-dler_t handler)	Installs a new signal handler for the signal with number *signum*. The signal handler is set to handler which may be a user specified function or a standard function as **SIG_IGN**
unsigned int alarm(-unsigned int seconds)	Causes the system to generate a SIGALRM signal for the process after the number of real-time seconds specified by *seconds* have elapsed.

By **Gorge97**
cheatography.com/gorge97/

Not published yet.
Last updated 10th July, 2022.