Edward Tischler

Term Project Part V
COP 4600, Fall 2015

I affirm that this project submission is solely the product of my own efforts and that I neither broke nor bent any academic honesty rules.
Edward Tischler, 12/1/2015

PAGE 1: TABLE OF CONTENTS

Section or Subsection(s)	Page #
→Section I: Answers to the “Standard Questions”	3
→Section II: Learning Experience	4
→Section III: Implementation Strategy	7
→Section IV:	10
→Section V:	22
Cop4600.c	23
Syscalls.master	45
Init_main.c	71
Kern_fork.c	98
Kern_exit.c	116
Proc.h	134
Cop4600.h	153
→Section VI:	156
→SectionVII:	158
→SectionVIII:	160
→SectionIX:	164

SECTION I: ANSWERS TO THE “STANDARD QUESTIONS”

1. Does the program compile without errors?
    - Yes

2. Does the program compile without warnings?
    - Yes

3. Does the program run without crashing?
    - Yes

4. Describe how you tested the program.
    - I tested the program by getting one syscall or action in the kernel working at a time. Therefore every time I completed an implementation I wrote an appropriate test case. If it worked, I moved on to the next syscall or action in the kernel.

5. Describe the ways in which the program not meet assignment specifications.
    - The program meets assignment specifications.
    -
6. Describe all known and suspected bugs.
    - I created a test case for every possible action in each syscall and kernel actions. So therefore I am not aware of nor do I suspect any bugs.

7. Does the program run correctly?
    - Yes

SECTION II: LEARNING EXPERIENCE

I believe the educational objective of this assignment was to get a student familiar with writing kernel code and to have him or her implement an already known concept (semaphores), which can be asked of us when we enter industry. However, this was the broad sense of the objective. In this assignment I learned many helpful practices that will help me later on. For example, I made the fatal mistake of not backing up my files. As a result, I lost the small amount of work that I had already done when I started. I quickly came to the conclusion that I needed to back up my files and have version control. As a result every time I compiled kernel code and booted the OS, if it booted correctly I saved a copy of the sys and kern folder. In addition I learned good techniques as in, compiling less frequently, looking for common mistakes before moving on, and to have a stagnated approached.

One of the biggest obstacles of this assignment was the fact that compiling took about ~3/4 minutes every time you wanted to recompile the kernel code and start the new OS with it. I am naturally a compile and test happy person. I like to compile whenever I write a syntactically complex line of code and test it to make sure it is functioning correctly. As a result it was hard to get off the ground in this assignment. This is because kernel code is much more different that user space code in that is has more arguments for common commands with higher complexity of the arguments. As a result, it would take multiple attempts at compiling to make sure the syntax was correct. Then once you got done compiling you needed to restart the OS which took time in itself. Finally there was a chance that you made a logical error or you wrote buggy code in the original kernel which would make you have a run time fault (during testing) or not boot correctly what so ever. Through this process I learned to compile less often and to check my code for errors that I have previously seen. Checking for errors helped me not run into runtime errors that usually occurred when using SIMPLEQs and accessing a NULL pointer. This also caused me to write my code one function at a time and make sure that it is accurate. It was easier to make sure that one function worked at a time and to make a test case for that function which would test every possible case for that function. Lastly, I believe I learned how to write code in C very well. Previous assignments required little to no specific knowledge to how C works. But through this assignment I can say I am confident in my ability to write C and can call it one of my primary languages.

I believe that this is a very well thought out assignment. I believe it completes all the educational objects that it sets out to complete. This assignment was also very interesting. I heavily appreciate that we got to edit and work closely with a real operating system. My only reservation was that all the previous assignments took not nearly as long as this one. I believe this assignment should be broken up into two parts. I believe it would have a higher success rate if this were the case.

SECTION III: IMPLEMENTATION STRATEGY

Layout of Narrative:

->What happens on process creation :

I first learned that the root process is created in init_main.c and every process after that is created in kern_fork.c. For this reason I placed a call to my function cop4600_pes_create and a pointer to the new “pes” in appropriate locations in these c files. “Pes” stands for process extension struct. To edit the OpenBSD source code as little as possible I simply put a pointer to this extension struct and initialized it (using my function) whenever the process was created. There was no way to specify processes that were used specifically for my testing program and therefore I had to create an extension for every process. For a reason that will be discussed later I had to also create an extension for the root process. Inside the cop4600_pes_create function I do a few things. First, I malloc the extension struct, then I initialize a SIMPLEQ which will hold the process’s semaphores, then I have a pointer backwards to the process and lastly I have an integer flag as to whether it should return ECONNABORTED.

->Allocate :

Upon the syscall to allocate a semaphore in the current process I first take in the name of the name using appropriate kernel code and malloc the semaphore whether or not it is to be created or not. I check to see if the name is too long. If it is too long I free the pointer and return the appropriate too long value. Afterwards, I check to see if there is already a semaphore with the same name in the current process. If there is a process with the same name I free the pointer to the semaphore and return the appropriate value. Otherwise, I complete the creation by initializing the lock associated with the semaphore, initializing the waiting queue in the semaphore, give it it’s instance variable values, and lastly insert it into the current processes semaphore queue.

->Down:

Upon the call to down a semaphore I first take in the name of the semaphore from user space into kernel space. Afterwards, I check current process’s semaphores. If the current process doesn’t have the semaphore I move onto inherited semaphores first working with the parent and then the grandparent and so on until the root process is reached. For this reason that I search all the way up the process tree for a semaphore which may not exist I had to create process extensions for all the processes otherwise I would accidently access a NULL pointer and crash the OS. I check to see if the PID ID is 1. If it is we have successfully searched up the tree to the root process and we can return the value for not finding the semaphore. However, if we find the semaphore in the current process or through inheritance we then decrement that semaphore. If the resulting count of that semaphore is negative we insert it into the waiting queue and put it to sleep based on the process extension’s address. When we wake up if the should_econn_abort flag was set in the process it returns the appropriate value. Otherwise the process will continue on. Everywhere where the count was accessed is inside a lock for semaphore mutual exclusion on values which can take time to update.

->Up:

Upon the call for up semaphore I first take in the name of the semaphore from user to kernel space. I do the same process in down looking for the semaphore in the current process and then the parents. If a semaphore is found I increment the account. If the resulting count is less than or equal to 0 I wake up the process waiting the longest in the queue and remove it from the queue. If the semaphore is not found I return the appropriate value. Everywhere were the count is accessed I have in an appropriate lock.

->Free:

Upon the call for free semaphore I first take in the name of the semaphore from user to kernel space. I do the same process described in down to look for the semaphore in the current process or an inherited semaphore in a parent process. If we don’t find the semaphore I return the appropriate value. If the semaphore is found I first set all the econnabored flags of the processes in the waiting queue to 1 and wake them up. Since their flags were set to one when they wake up in down they will be caught by an if-statement and return that value. Afterwards I pop all the processes from the waiting queue. Once that is done I remove the semaphore from the queue.

->What happens on process exit:

Processes exit in kern_exit.c. I call my destroy cop4600_pes_destroy function. In this function I recursively do what free does without checking for the name. I simply see if the queue of semaphores is empty. If it is not empty I wake up all the processes in the waiting queue and have them return ECONNABORED. If there are no more semaphores then finish the process by calling free on the “pes” pointer. If it is not then recursively call the same function and destroy the next semaphore. Eventually all semaphores will be destroyed and the processes will exit.

SECTION IV: PART5.TXT

```
Script started on Fri Nov 27 12:11:58 2015
# sh

#

#

#

# cd /usr/src/sys/kern

# ls

#

#

# ls -lt | head

total 3084
-rw-r--r--  1 root  wsrc  14597 Dec  1 12:51 cop4600.c
-rw-r--r--  1 root  wsrc  16133 Nov 29 18:38 init_main.c
-rw-r--r--  1 root  wsrc  11149 Nov 29 18:38 kern_fork.c
-rw-r--r--  1 root  wsrc  52112 Nov 29 16:57 vfs_subr.c
-rw-r--r--  1 root  wsrc  11616 Nov 29 16:27 kern_proc.c
-rw-r--r--  1 root  wsrc  26582 Nov 26 21:07 init_sysent.c
-rw-r--r--  1 root  wsrc  15675 Nov 26 21:07 syscalls.c
-rw-r--r--  1 root  wsrc  14138 Nov 26 14:52 kern_exit.c
-rw-r--r--  1 root  wsrc  22359 Nov 25 23:01 syscalls.master
```

```
#

#

#

# tail syscalls.master

;==============================================================================


;added by Dave Small
289     STD             { int sys_hello( void ); }
290     STD             { int sys_showargs( const char *str, int val ); }
; added by Edward Tischler (Part 5 only)
291 STD         { int sys_allocate_semaphore( const char* name, int initial_count ); }
292 STD         { int sys_down_semaphore( const char* name ); }
293 STD         { int sys_up_semaphore( const char* name );}
294 STD         { int sys_free_semaphore( const char* name );}#

#

#

# grep semaphore *

cop4600.c:  /*pool_init(&sema_pool, sizeof(struct cop4600_sema), 0, 0, 0, "cop4600semapl",
//this will create the pool to hold the structs for semaphores
cop4600.c:  q holding semaphores belonging to this process
cop4600.c:  /*pool_init(&pes_pool, sizeof(struct cop4600_pes), 0, 0, 0, "cop4600pespl",  //this
will create the pool to hold the structs for semaphores
```

```
cop4600.c:          //uprintf("processes being removed from wait queue when semaphore freed from proc exit\n");
cop4600.c:    //uprintf("semaphore freed from a process upon process exit \n");
cop4600.c:int sys_allocate_semaphore(struct proc *p, void *v, register_t *retval) {
cop4600.c:  struct sys_allocate_semaphore_args *uap = v;
cop4600.c://check to see if there is already a semaphore with that name
cop4600.c:  //uprintf("semaphore created in a process \n");
cop4600.c:int sys_down_semaphore(struct proc *p, void *v, register_t *retval) {
cop4600.c:int foundsemaphore = 0;
cop4600.c:struct sys_allocate_semaphore_args *uap = v;
cop4600.c://first begin with seeing if the semaphore exists
cop4600.c:        foundsemaphore = 1;
cop4600.c:        //uprintf("new semaphore count: %i \n", np->count);
cop4600.c:          //uprintf("name of semaphore going to sleep: %s\n", np->name);
cop4600.c:    if(foundsemaphore == 0){
cop4600.c:  if(foundsemaphore == 0){
cop4600.c:    //uprintf("semaphore not found for down\n");
cop4600.c:int sys_up_semaphore(struct proc *p, void *v, register_t *retval) {
cop4600.c:int foundsemaphore = 0;
cop4600.c:struct sys_allocate_semaphore_args *uap = v;
cop4600.c://first begin with seeing if the semaphore exists
cop4600.c:        foundsemaphore = 1;
cop4600.c:        //uprintf("new semaphore count: %i \n", np->count);
cop4600.c:    if(foundsemaphore == 0){
cop4600.c:  if(foundsemaphore == 0){
cop4600.c:    //uprintf("semaphore not found in up\n");
cop4600.c:int sys_free_semaphore(struct proc *p, void *v, register_t *retval) {
```

```
cop4600.c:int foundsemaphore = 0;
cop4600.c:struct sys_allocate_semaphore_args *uap = v;
cop4600.c://first begin with seeing if the semaphore exists
cop4600.c:       foundsemaphore = 1;
cop4600.c:         //uprintf("processes being removed from wait queue when semaphore freed\n");
cop4600.c:    if(foundsemaphore == 0){
cop4600.c:      if(foundsemaphore == 1){
cop4600.c:       // //uprintf("semaphore freed count should be 0\n");
cop4600.c:////uprintf("number of semaphores after free: %i\n", i);
cop4600.c:  if(foundsemaphore == 0){
cop4600.c:    //uprintf("semaphore not found in free semaphore\n");
init_main.c:    /* Initialize System V style semaphores. */
init_sysent.c:  { 2, s(struct sys_allocate_semaphore_args),
init_sysent.c:      sys_allocate_semaphore },          /* 291 = allocate_semaphore */
init_sysent.c:  { 1, s(struct sys_down_semaphore_args),
init_sysent.c:      sys_down_semaphore },              /* 292 = down_semaphore */
init_sysent.c:  { 1, s(struct sys_up_semaphore_args),
init_sysent.c:      sys_up_semaphore },                /* 293 = up_semaphore */
init_sysent.c:  { 1, s(struct sys_free_semaphore_args),
init_sysent.c:      sys_free_semaphore },              /* 294 = free_semaphore */
syscalls.c:     "allocate_semaphore",                /* 291 = allocate_semaphore */
syscalls.c:     "down_semaphore",                    /* 292 = down_semaphore */
syscalls.c:     "up_semaphore",                /* 293 = up_semaphore */
syscalls.c:     "free_semaphore",                    /* 294 = free_semaphore */
syscalls.master:291 STD        { int sys_allocate_semaphore( const char* name, int initial_count ); }
syscalls.master:292 STD        { int sys_down_semaphore( const char* name ); }
syscalls.master:293 STD         { int sys_up_semaphore( const char* name );}

syscalls.master:294 STD         { int sys_free_semaphore( const char* name );}

sysv_sem.c: * Implementation of SVID semaphores

sysv_sem.c:struct       semid_ds **sema;        /* semaphore id list */

sysv_sem.c:      * Preallocate space for the new semaphore.  If we are going

sysv_sem.c:      * condition in allocating a semaphore with a specific key.

sysv_sem.c:                     DPRINTF(("not enough semaphores left (need %d, got %d)\n",

sysv_sem.c:             * Make sure that the semaphore still exists

sysv_sem.c:             * The semaphore is still alive.  Readjust the count of

sysv_sem.c:                      * rollback the semaphore ups and down so we can return

sysv_sem.c:     /* Do a wakeup if any semaphore was up'd. */

sysv_sem.c: * semaphores.

#


#


#


# cd ../arch/i386/compiule               le/GENC     ERIC


#


#


#


# grep semaphore *


Binary file cop4600.o matches

Binary file init_sysent.o matches
```

```
param.c: * Values in support of System V compatible semaphores.
param.c:        SEMMNI,         /* # of semaphore identifiers */
param.c:        SEMMNS,         /* # of semaphores in system */
param.c:        SEMMSL,         /* max # of semaphores per id */
param.c:        SEMVMX,         /* semaphore maximum value */
#

#

#

# grep cop4600 MA     akefile

OBJS=   cop4600.o \
CFILES= $S/kern/cop4600.c \
#

#

#

# tail Makefile


uscanner.o: $S/dev/usb/uscanner.c
        ${NORMAL_C}


usscanner.o: $S/dev/usb/usscanner.c
        ${NORMAL_C}
```

```
if_wi_usb.o: $S/dev/usb/if_wi_usb.c
        ${NORMAL_C}


#

#

#

# cd /root

#

#

#

# ls -lt

total 144
-rw-r--r--  1 root  wheel  1491 Dec  1 13:10 firsttestfile.c
-rw-r--r--  1 root  wheel  6689 Nov 27 12:13 part5.txt
-rwxr-xr-x  1 root  wheel  7192 Nov 27 12:04 test
drwx------  2 root  wheel   512 Nov 27 11:57 .vnc
-rw-------  1 root  wheel   333 Nov 27 11:57 .Xauthority
-rwxr-xr-x  1 root  wheel  7032 Nov 27 11:54 a.out
-rw-r--r--  1 root  wheel   138 Nov 27 11:04 output.txt
-rwxr-xr-x  1 root  wheel  6520 Nov 26 23:55 tesyt
-rw-r--r--  1 root  wheel     0 Nov 25 19:02 firsttestfile.c~
-rw-r--r--  1 root  wheel   676 Jun  9  2006 kerntest.c
```

```
-rwxr-xr-x  1 root  wheel  6536 Jun  9  2006 kerntest

drwx------  2 root  wheel   512 Jun  9  2006 .ssh

-rwxr-xr-x  1 root  wheel    67 Jun  9  2006 vncup800

-rwxr-xr-x  1 root  wheel    68 Jun  9  2006 vncup1024

-rwxr-xr-x  1 root  wheel    69 Jun  9  2006 vncup1280

-rw-r--r--  1 root  wheel   633 Jun  9  2006 .fonts.cache-1

drwxr-xr-x  3 root  wheel   512 Jun  9  2006 .emacs.d

-rw-r--r--  1 root  wheel   482 Jun  9  2006 .emacs

-rw-r--r--  2 root  wheel   769 Jun  9  2006 .cshrc

-rw-r--r--  2 root  wheel   267 Jun  9  2006 .profile

-rw-------  1 root  wheel   125 Mar 29  2004 .klogin

-rw-r--r--  1 root  wheel   335 Mar 29  2004 .login

#


#


#


# gcc -o semtest f
# gcc -o semtest firsttestfile.c

# gcc -o semtest firsttestfile.c


#


#


#


# ./semtest
```

```
Starting 3 process test
Testing up and down
Semaphore allocated
Parent downing semaphore
Count still not negative downing again (should sleep)
Child up-ing semaphore owned by parent
Other child up-ing non existant semaphore
process finished 1st test
process finished 1st test
process finished 1st test
Starting free test(2nd test)
Parent downing semaphore to sleep
Process finished 2nd test
Child freeing parent's semaphore
Process finished 2nd test
Process finished 2nd test
Process Finished Test
Process Finished Test
Starting automatic destruction test (parent only)
Allocating semaphore
Results can't be seen but if there is no fault then semaphore destroyed
Process Finished Test
#

#

#
```

```
# ls -lt \     /

total 20080
drwx------   5 root  wheel      512 Nov 27 12:15 root
drwxrwxrwt   4 root  wheel      512 Nov 27 12:14 tmp
drwxr-xr-x   3 root  wheel    19968 Nov 27 11:57 dev
-rwxr-xr-x   1 root  wsrc   5077491 Nov 27 11:46 bsd
lrwxr-xr-x   1 root  wheel       16 Nov 27 07:09 shortcuttokern -> usr/src/sys/kern
drwxr-xr-x  18 root  wheel     2048 May 20  2008 etc
-rw-r--r--   2 root  wheel      769 Jun  9  2006 .cshrc
-rw-r--r--   2 root  wheel      267 Jun  9  2006 .profile
-rw-r--r--   1 root  wheel  5075323 Jul  7  2005 bsd.0
-rw-r--r--   1 root  wheel    42132 Jul 18  2004 boot
lrwxr-xr-x   1 root  wheel       11 Jul 18  2004 sys -> usr/src/sys
drwxr-xr-x   2 root  wheel     2048 Mar 29  2004 sbin
drwxr-xr-x   2 root  wheel     1024 Mar 29  2004 bin
drwxr-xr-x   2 root  wheel      512 Mar 29  2004 altroot
drwxr-xr-x   2 root  wheel      512 Mar 29  2004 home
drwxr-xr-x   2 root  wheel      512 Mar 29  2004 mnt
drwxr-xr-x   2 root  wheel      512 Mar 29  2004 stand
drwxr-xr-x  22 root  wheel      512 May 16  2003 var
drwxr-xr-x  16 root  wheel      512 May 16  2003 usr
#

#

#
```

```
# dmesg | head

2 head, 18 sec
biomask 4a40 netmask 4e40 ttymask 5ec2
pctr: 686-class user-level performance counters enabled
mtrr: Pentium Pro MTRR support
dkcsum: sd0 matched BIOS disk 80
dkcsum: sd1 matched BIOS disk 81
root on sd0a
rootdev=0x400 rrootdev=0xd00 rawdev=0xd02
WARNING: / was not properly unmounted
syncing disks... done
# #
Script done on Fri Nov 27 12:15:55 2015
```

SECTION V: NOVEL AND MODIFIED SOURCE CODE

COP4600.C

USR/SRC/SYS/KERN/COP4600.C

```
/*                                      $OpenBSD: cop4600.c,v 1.00 2003/07/12 01:33:27 dts Exp $
                                        */

#include <sys/param.h>
#include <sys/acct.h>
#include <sys/systm.h>
#include <sys/ucred.h>
#include <sys/proc.h>
#include <sys/timeb.h>
#include <sys/times.h>
#include <sys/malloc.h>
#include <sys/filedesc.h>
#include <sys/pool.h>

#include <sys/mount.h>
#include <sys/syscallargs.h>

//added by ed
#include <sys/cop4600.h>
#include <sys/queue.h>
#include <sys/types.h>
#include <sys/lock.h>

/*======================================================================**
**  Dave's example system calls                                         **
**======================================================================*/
```

```
/*
** hello() uprints to the tty a hello message and returns the process id
*/

int
sys_hello( struct proc *p, void *v, register_t *retval )
{
  //uprintf( "\nHello, process %d!\n", p->p_pid );

  //uprintf("number of processes: %i \n", nprocs);

  *retval = p->p_pid;

  return (0);
}

/*
** showargs() demonstrates passing arguments to the kernel
*/

#define MAX_STR_LENGTH  1024

int
sys_showargs( struct proc *p, void *v, register_t *retval )
{
  /* The arguments are passed in a structure defined as:
```

```
  **
  **  struct sys_showargs_args
  **  {
  **      syscallarg(char *) str;
  **      syscallarg(int)    val;
  **  }
  */

  struct sys_showargs_args *uap = v;

  char kstr[MAX_STR_LENGTH+1]; /* will hold kernal-space copy of uap->str */
  int err = 0;
  int size = 0;

  /* copy the user-space arg string to kernal-space */

  err = copyinstr( SCARG(uap, str), &kstr, MAX_STR_LENGTH, &size );
  if (err == EFAULT)
    return( err );

  //uprintf( "The argument string is \"%s\"\n", kstr );
  //uprintf( "The argument integer is %d\n", SCARG(uap, val) );
  *retval = 0;

  return (0);
}
```

```
/*=========================================================================**
**  <Edward Tischler>'s COP4600 20015C system calls                    **
**==========================================================================*/
//extra structs for part 5
//this is needed to set up the pools
//struct pool sema_pool;
//struct pool pes_pool;



struct cop4600_sema {

  SIMPLEQ_ENTRY(cop4600_sema) sema_entries;

  char name[32];
  int count;

  struct simplelock myslock;
  /* todo
  q holding waiting process ids that will be woken up as count goes non negative
  add lock
  */

  SIMPLEQ_HEAD(otherlisthead,cop4600_pes) otherhead; //must be proc to hold the process
};

//TODO add a sema create
                        //maybe?
```

```
//TODO add a sema destroy

int cop4600_sema_init() {

  /*pool_init(&sema_pool, sizeof(struct cop4600_sema), 0, 0, 0, "cop4600semapl", //this will
create the pool to hold the structs for semaphores
      &pool_allocator_nointr);*/

  return 0;
}



struct cop4600_pes {

  SIMPLEQ_ENTRY(cop4600_pes) proc_entries;

  /*todo
  q holding semaphores belonging to this process
  */
  SIMPLEQ_HEAD(listhead,cop4600_sema) head;

  struct proc* thisprocess;

  int should_econn_abort;



};
```

```
int cop4600_pes_init() {

  /*pool_init(&pes_pool, sizeof(struct cop4600_pes), 0, 0, 0, "cop4600pespl",  //this will create the pool to hold the structs for semaphores
      &pool_allocator_nointr);*/



  return 0;
}


int cop4600_pes_create(struct cop4600_pes **pes, struct proc *p) {

  /**pes = pool_get(&pes_pool, PR_WAITOK);*/
 //
  ////uprintf("process create called\n");

  *pes = malloc(sizeof(struct cop4600_pes), M_TEMP, M_NOWAIT);
  SIMPLEQ_INIT(&(*pes)->head);
  (*pes)->thisprocess = p;
  (*pes)->should_econn_abort = 0;
  //(*pes)->myval = 2;

  return 0;
```

```
}


int cop4600_pes_destroy(struct cop4600_pes **pes) {
  // TODO: Destroy things internal to pes

  // Put the destroyed process extension back in the pool.
  //pool_put(&pes_pool, *pes);
  // Process extension no longer assigned.
  //pid_t curprocess = (*pes)->thisprocess->p_pid;
  struct cop4600_pes *waiting_queuenodes;
  struct cop4600_sema *np;
  //struct cop4600_sema *othernp;

  np = SIMPLEQ_FIRST(&((*pes)->head));


  if(np!=NULL){
    for(waiting_queuenodes = SIMPLEQ_FIRST(&(np->otherhead));waiting_queuenodes!=NULL;waiting_queuenodes = SIMPLEQ_NEXT(waiting_queuenodes,proc_entries)){

      waiting_queuenodes->should_econn_abort = 1;
      wakeup((waiting_queuenodes));
      //uprintf("process woken up and returned from free on proc exit\n");

    }
  }
  if(np!=NULL){
    while(! SIMPLEQ_EMPTY( &((np)->otherhead) )){
          //uprintf("processes being removed from wait queue when semaphore freed from proc exit\n");
          SIMPLEQ_REMOVE_HEAD( &((np)->otherhead), (waiting_queuenodes = SIMPLEQ_FIRST(&((np)->otherhead))), proc_entries );



     }
   }




if(! SIMPLEQ_EMPTY( &((*pes)->head))){
SIMPLEQ_REMOVE_HEAD( &((*pes)->head), (np = SIMPLEQ_FIRST(&((*pes)->head))), sema_entries );
   free(np, M_TEMP);
   //uprintf("semaphore freed from a process upon process exit \n");
}




if(! SIMPLEQ_EMPTY( &((*pes)->head))){
  cop4600_pes_destroy((pes));
}
```

```
////uprintf("9\n");

else{

  free(*pes,M_TEMP);

  ////uprintf("10\n");

  *pes = NULL;

}

  return 0;

}



int sys_allocate_semaphore(struct proc *p, void *v, register_t *retval) {


  struct sys_allocate_semaphore_args *uap = v;


  char kstr[MAX_STR_LENGTH+1]; /* will hold kernal-space copy of uap->str */

  int err = 0;

  int size = 0;

  int samename = 0;


  struct cop4600_sema *np;


  struct cop4600_sema *sema_pointer = /*pool_get(&sema_pool, PR_WAITOK);*/
malloc(sizeof(struct cop4600_sema), M_TEMP, M_NOWAIT);

  /* copy the user-space arg string to kernal-space */


  err = copyinstr( SCARG(uap, name), &kstr, MAX_STR_LENGTH, &size );
  if (err == EFAULT)
    return( err );

if(strlen(kstr) > 31){
  free(sema_pointer, M_TEMP);
  return (ENAMETOOLONG);
}

else{

//check to see if there is already a semaphore with that name
for(np = SIMPLEQ_FIRST(&(p->pes->head));np!=NULL;np = SIMPLEQ_NEXT(np,sema_entries)){
  if(strcmp(kstr, np->name) == 0){
    //uprintf("same name sorry\n");
    samename = 1;
    free(sema_pointer, M_TEMP);
    return (EEXIST);
  }
}

  //add characteristics
//WONT GET HERE IF SAME NAME BECAUSE OF PREVIOUS RETURN
  strncpy(sema_pointer->name, kstr, 32);
  sema_pointer->name[31] = '\0';
  sema_pointer->count = SCARG(uap, initial_count);
  //sema_pointer->lock = malloc(sizeof(struct myslock), M_TEMP, M_NOWAIT); NOT ACCURATE
  simple_lock_init(&(sema_pointer->myslock));
  SIMPLEQ_INIT(&(sema_pointer)->otherhead);
  //uprintf("semaphore created in a process \n");
  SIMPLEQ_INSERT_TAIL(&(p->pes->head), sema_pointer, sema_entries);
}


  /*for(np = SIMPLEQ_FIRST(&(p->pes->head));np!=NULL; np = 
SIMPLEQ_NEXT(np,sema_entries)){
    //uprintf("this is the name: %s", np->name);
    //uprintf("\n");
    ////uprintf("%i",size);
  }*/


  *retval = 0;
  return (0);
}


int sys_down_semaphore(struct proc *p, void *v, register_t *retval) {
int parentend = 0;
int foundsemaphore = 0;
struct cop4600_sema *np;
struct proc *pcheck;
//int i = 0;

struct sys_allocate_semaphore_args *uap = v;
```

```
  char kstr[MAX_STR_LENGTH+1]; /* will hold kernal-space copy of uap->str */
  int err = 0;
  int size = 0;
 // pid_t currentpid;
  //int i = 0;

   err = copyinstr( SCARG(uap, name), &kstr, MAX_STR_LENGTH, &size );
  if (err == EFAULT)
    return( err );
pcheck = p;
//first begin with seeing if the semaphore exists
  while(parentend == 0){
//for( i = 0; i < 35; i++){
    for(np = SIMPLEQ_FIRST(&(pcheck->pes->head));np!=NULL;np =
SIMPLEQ_NEXT(np,sema_entries)){

      if(strcmp(kstr, np->name) == 0){

        //need to start lock
        simple_lock(&(np->myslock));
        np->count -= 1;
        foundsemaphore = 1;
        parentend = 1; //this will ensure it does not try to search the parent
        //uprintf("new semaphore count: %i \n", np->count);
        //free(sema_pointer, M_TEMP);
        //now if it is negative I will need to add this process to the waiting queue
        if(np->count < 0){
```

```
      //SIMPLEQ_INSERT_TAIL(&(p->pes->head), sema_pointer, sema_entries);  ------ for reference


      SIMPLEQ_INSERT_TAIL(&(np->otherhead), p->pes , proc_entries);




      //make process wait now
      //uprintf("putting to sleep\n");
      //uprintf("name of semaphore going to sleep: %s\n", np->name);
     // //uprintf("value of tsleep address %p\n", (p->pes));
      tsleep((p->pes),0, "tsleeping" , 0);
      ////uprintf("value of tsleep address %s\n", &(p->pes));


      //wakeup(p);
      //uprintf("process has been woken up\n");


      if(p->pes->should_econn_abort == 1){
        p->pes->should_econn_abort = 0;
        return (ECONNABORTED);
      }


      ////uprintf("this happens");
      simple_unlock(&(np->myslock));
      //need to end lock
    }
  }
  }
  //currentpid = getpid();
  if(foundsemaphore == 0){
   if(pcheck->p_pid == 1){
    ////uprintf("1\n");
    parentend = 1; //this will prevent faulting
   }
   else{
   // //uprintf("2\n");
    pcheck = pcheck->p_pptr; //will change process to parent to find
   }
  }

 }

 if(foundsemaphore == 0){
  //uprintf("semaphore not found for down\n");
  return (ENOENT);
 }



 *retval = 0;
 return (0);
}
```

```
int sys_up_semaphore(struct proc *p, void *v, register_t *retval) {
  // TODO
int parentend = 0;
int foundsemaphore = 0;
struct cop4600_sema *np;

struct cop4600_pes *waiting_queuenodes;
struct proc *pcheck;
struct sys_allocate_semaphore_args *uap = v;
  char kstr[MAX_STR_LENGTH+1]; /* will hold kernal-space copy of uap->str */
  int err = 0;
  int size = 0;
 // pid_t currentpid;
  //int i = 0;

   err = copyinstr( SCARG(uap, name), &kstr, MAX_STR_LENGTH, &size );
  if (err == EFAULT)
    return( err );
pcheck = p;
//first begin with seeing if the semaphore exists
  while(parentend == 0){
//for( i = 0; i < 35; i++){
    for(np = SIMPLEQ_FIRST(&(pcheck->pes->head));np!=NULL;np =
SIMPLEQ_NEXT(np,sema_entries)){

      if(strcmp(kstr, np->name) == 0){
```

```
    /*for(waiting_queuenodes = SIMPLEQ_FIRST(&(np-
>otherhead));waiting_queuenodes!=NULL;waiting_queuenodes =
SIMPLEQ_NEXT(waiting_queuenodes,proc_entries)){

     //uprintf("addresses in the queue for wakeup: %p\n", waiting_queuenodes);

    }*/

    //need to start lock

    simple_lock(&(np->myslock));

    np->count += 1;

    foundsemaphore = 1;

    parentend = 1; //this will ensure it does not try to search the parent

    //uprintf("new semaphore count: %i \n", np->count);

    //free(sema_pointer, M_TEMP);

    //now if it is negative I will need to add this process to the waiting queue

    if(np->count <= 0){

     ////uprintf("first\n");

     if(!SIMPLEQ_EMPTY(&(np->otherhead)) ){ //if there is a waiting process wake it up fifo

      ////uprintf("sencond\n");

      wakeup((SIMPLEQ_FIRST(&(np->otherhead))));

      ////uprintf("wake up address %p\n", (SIMPLEQ_FIRST(&(np->otherhead))));

      //TODO

     // //uprintf("thrid\n");

        SIMPLEQ_REMOVE_HEAD( &((np)->otherhead), (waiting_queuenodes =
SIMPLEQ_FIRST(&((np)->otherhead))), proc_entries );

       // SIMPLEQ_REMOVE_HEAD( &((*pes)->head), (np = SIMPLEQ_FIRST(&((*pes)-
>head))), sema_entries ); ----- for reference

        ////uprintf("fourth\n");

       simple_unlock(&(np->myslock));

     }
     else{
      //uprintf("wait queue is empty\n");
     }

    }
   }

  }
  //currentpid = getpid();
  if(foundsemaphore == 0){
   if(pcheck->p_pid == 1){
    ////uprintf("1\n");
    parentend = 1; //this will prevent faulting
   }
   else{
   // //uprintf("2\n");
    pcheck = pcheck->p_pptr; //will change process to parent to find
   }
  }

 }

 if(foundsemaphore == 0){
  //uprintf("semaphore not found in up\n");
  return (ENOENT);
 }
```

```
  *retval = 0;
  return (0);
}

int sys_free_semaphore(struct proc *p, void *v, register_t *retval) {

int parentend = 0;
int foundsemaphore = 0;
struct cop4600_sema *np;
struct cop4600_sema *othernp;
struct proc *pcheck;
// curprocess = p->p_pid;

struct cop4600_pes *waiting_queuenodes;

struct sys_allocate_semaphore_args *uap = v;
  char kstr[MAX_STR_LENGTH+1]; /* will hold kernal-space copy of uap->str */
  int err = 0;
  int size = 0;
 // pid_t currentpid;
  //int i = 0;
```

```
   err = copyinstr( SCARG(uap, name), &kstr, MAX_STR_LENGTH, &size );
  if (err == EFAULT)
    return( err );
pcheck = p;
//first begin with seeing if the semaphore exists
  while(parentend == 0){
//for( i = 0; i < 35; i++){
    for(np = SIMPLEQ_FIRST(&(pcheck->pes->head));np!=NULL;np =
SIMPLEQ_NEXT(np,sema_entries)){


      if(strcmp(kstr, np->name) == 0){


        simple_lock(&(np->myslock));
        othernp = np;
        parentend = 1;
        foundsemaphore = 1;


        for(waiting_queuenodes = SIMPLEQ_FIRST(&(np-
>otherhead));waiting_queuenodes!=NULL;waiting_queuenodes =
SIMPLEQ_NEXT(waiting_queuenodes,proc_entries)){


        // //uprintf("wake up called. count this for number of processes in wait queue, address
%p\n", waiting_queuenodes);
         waiting_queuenodes->should_econn_abort = 1;
          wakeup((waiting_queuenodes));


          //if(waiting_queuenodes->thisprocess->p_pid != curprocess){
            //uprintf("process woken up and returned from free\n");
      //return (ECONNABORTED);
     //}

    }

    while(! SIMPLEQ_EMPTY( &((np)->otherhead) )){
     //uprintf("processes being removed from wait queue when semaphore freed\n");
     SIMPLEQ_REMOVE_HEAD( &((np)->otherhead), (waiting_queuenodes =
SIMPLEQ_FIRST(&((np)->otherhead))), proc_entries );

    }

    simple_unlock(&(np->myslock));

   }
  }

  if(foundsemaphore == 0){
   if(pcheck->p_pid == 1){
    ////uprintf("1\n");
    parentend = 1; //this will prevent faulting
   }
   else{
    ////uprintf("2\n");
    pcheck = pcheck->p_pptr; //will change process to parent to find
   }
  }
```

```
   if(foundsemaphore == 1){
    // //uprintf("semaphore freed count should be 0\n");
     SIMPLEQ_REMOVE_HEAD( &(pcheck->pes->head), SIMPLEQ_FIRST(&(pcheck->pes->head)),
sema_entries );
     free(othernp, M_TEMP);
     //othernp = SIMPLEQ_NEXT()
   }



 }
  //currentpid = getpid();




////uprintf("number of semaphores after free: %i\n", i);
 if(foundsemaphore == 0){
  //uprintf("semaphore not found in free semaphore\n");
  return (ENOENT);
 }






 *retval = 0;
 return (0);
}
```

SYSCALLS.MASTER

USR/SRC/SYS/KERN/SYSCALLS.MASTER

```
;                                   $OpenBSD: syscalls.master,v 1.68 2004/02/28 19:44:16 miod
Exp $
;                                   $NetBSD: syscalls.master,v 1.32 1996/04/23 10:24:21
mycroft Exp $


;                                   @(#)syscalls.master     8.2 (Berkeley) 1/13/94


; OpenBSD system call name/number "master" file.
; (See syscalls.conf to see what it is processed into.)
;
; Fields: number type [type-dependent ...]
;                                   number  system call number, must be in order
;                                   type    one of STD, OBSOL, UNIMPL, NODEF, NOARGS, or
one of
;                                     the compatibility options defined in syscalls.conf.
;
; types:
;                                   STD     always included
;                                   OBSOL   obsolete, not included in system
;                                   UNIMPL  unimplemented, not included in system
;                                   NODEF   included, but don't define the syscall number
;                                   NOARGS  included, but don't define the syscall args structure
;                                   INDIR   included, but don't define the syscall args
structure,
;                                     and allow it to be "really" varargs.
;
; The compat options are defined in the syscalls.conf file, and the
; compat option name is prefixed to the syscall name.  Other than
; that, they're like NODEF (for 'compat' options), or STD (for
; 'libcompat' options).
;
; The type-dependent arguments are as follows:
; For STD, NODEF, NOARGS, and compat syscalls:
;                       { pseudo-proto } [alias]
; For other syscalls:
;                       [comment]
;
; #ifdef's, etc. may be included, and are copied to the output files.
; #include's are copied to the syscall switch definition file only.

#include <sys/param.h>
#include <sys/systm.h>
#include <sys/signal.h>
#include <sys/mount.h>
#include <sys/syscallargs.h>
#include <sys/poll.h>
#include <sys/event.h>
#include <xfs/xfs_pioctl.h>

; Reserved/unimplemented system calls in the range 0-150 inclusive
; are reserved for use in future Berkeley releases.
; Additional system calls implemented in vendor and other
; redistributions should be placed in the reserved range at the end
; of the current calls.
```

```
0                        INDIR          { int sys_syscall(int number, ...); }
1                        STD            { void sys_exit(int rval); }
2                        STD            { int sys_fork(void); }
3                        STD            { ssize_t sys_read(int fd, void *buf, size_t
nbyte); }
4                        STD            { ssize_t sys_write(int fd, const void *buf, \
                                    size_t nbyte); }
5                        STD            { int sys_open(const char *path, \
                                    int flags, ... int mode); }
6                        STD            { int sys_close(int fd); }
7                        STD            { pid_t sys_wait4(pid_t pid, int *status, int
options, \
                                    struct rusage *rusage); }
8                        COMPAT_43      { int sys_creat(const char *path, int mode);
} ocreat
9                        STD            { int sys_link(const char *path, const char
*link); }
10                       STD            { int sys_unlink(const char *path); }
11                       OBSOL          execv
12                       STD            { int sys_chdir(const char *path); }
13                       STD            { int sys_fchdir(int fd); }
14                       STD            { int sys_mknod(const char *path, int mode,
\
                                    dev_t dev); }
15                       STD            { int sys_chmod(const char *path, int
mode); }
16                       STD            { int sys_chown(const char *path, uid_t uid,
\
                                        gid_t gid); }
17                           STD            { int sys_obreak(char *nsize); } break
18                           COMPAT_25      { int sys_getfsstat(struct statfs *buf, long
bufsize, \
                                        int flags); } ogetfsstat
19                           COMPAT_43      { long sys_lseek(int fd, long offset, int
whence); } \
                                        olseek
20                           STD            { pid_t sys_getpid(void); }
21                           STD            { int sys_mount(const char *type, const
char *path, \
                                        int flags, void *data); }
22                           STD            { int sys_unmount(const char *path, int
flags); }
23                           STD            { int sys_setuid(uid_t uid); }
24                           STD            { uid_t sys_getuid(void); }
25                           STD            { uid_t sys_geteuid(void); }
#ifdef PTRACE
26                           STD            { int sys_ptrace(int req, pid_t pid, caddr_t
addr, \
                                        int data); }
#else
26                           UNIMPL         ptrace
#endif
27                           STD            { ssize_t sys_recvmsg(int s, struct msghdr
*msg, \
                                        int flags); }
28                           STD            { ssize_t sys_sendmsg(int s, \
                                        const struct msghdr *msg, int flags); }
```

```
29                      STD             { ssize_t sys_recvfrom(int s, void *buf,
size_t len, \
                                    int flags, struct sockaddr *from, \
                                    socklen_t *fromlenaddr); }
30                      STD             { int sys_accept(int s, struct sockaddr
*name, \
                                    socklen_t *anamelen); }
31                      STD             { int sys_getpeername(int fdes, struct
sockaddr *asa, \
                                    int *alen); }
32                      STD             { int sys_getsockname(int fdes, struct
sockaddr *asa, \
                                    socklen_t *alen); }
33                      STD             { int sys_access(const char *path, int flags);
}
34                      STD             { int sys_chflags(const char *path, u_int
flags); }
35                      STD             { int sys_fchflags(int fd, u_int flags); }
36                      STD             { void sys_sync(void); }
37                      STD             { int sys_kill(int pid, int signum); }
38                      COMPAT_43       { int sys_stat(const char *path, struct ostat
*ub); } \
                                    ostat
39                      STD             { pid_t sys_getppid(void); }
40                      COMPAT_43       { int sys_lstat(char *path, \
                                    struct ostat *ub); } olstat
41                      STD             { int sys_dup(int fd); }
42                      STD             { int sys_opipe(void); }
43                      STD             { gid_t sys_getegid(void); }
```

```
44                       STD             { int sys_profil(caddr_t samples, size_t size,
\
                                     u_long offset, u_int scale); }
#ifdef KTRACE
45                       STD             { int sys_ktrace(const char *fname, int ops,
\
                                     int facs, pid_t pid); }
#else
45                       UNIMPL          ktrace
#endif
46                       STD             { int sys_sigaction(int signum, \
                                     const struct sigaction *nsa, \
                                     struct sigaction *osa); }
47                       STD             { gid_t sys_getgid(void); }
48                       STD             { int sys_sigprocmask(int how, sigset_t
mask); }
49                       STD             { int sys_getlogin(char *namebuf, u_int
namelen); }
50                       STD             { int sys_setlogin(const char *namebuf); }
51                       STD             { int sys_acct(const char *path); }
52                       STD             { int sys_sigpending(void); }
53                       STD             { int sys_osigaltstack(const struct
osigaltstack *nss, \
                                     struct osigaltstack *oss); }
54                       STD             { int sys_ioctl(int fd, \
                                     u_long com, ... void *data); }
55                       STD             { int sys_reboot(int opt); }
56                       STD             { int sys_revoke(const char *path); }
57                       STD             { int sys_symlink(const char *path, \
                                        const char *link); }
58                      STD             { int sys_readlink(const char *path, char
*buf, \
                                        size_t count); }
59                      STD             { int sys_execve(const char *path, \
                                        char * const *argp, char * const *envp); }
60                      STD             { int sys_umask(int newmask); }
61                      STD             { int sys_chroot(const char *path); }
62                      COMPAT_43       { int sys_fstat(int fd, struct ostat *sb); }
ofstat
63                      COMPAT_43       { int sys_getkerninfo(int op, char *where,
int *size, \
                                        int arg); } ogetkerninfo
64                      COMPAT_43       { int sys_getpagesize(void); } ogetpagesize
65                      COMPAT_25       { int sys_omsync(caddr_t addr, size_t len); }
66                      STD             { int sys_vfork(void); }
67                      OBSOL           vread
68                      OBSOL           vwrite
69                      STD             { int sys_sbrk(int incr); }
70                      STD             { int sys_sstk(int incr); }
71                      COMPAT_43       { int sys_mmap(caddr_t addr, size_t len, int
prot, \
                                        int flags, int fd, long pos); } ommap
72                      STD             { int sys_ovadvise(int anom); } vadvise
73                      STD             { int sys_munmap(void *addr, size_t len); }
74                      STD             { int sys_mprotect(void *addr, size_t len, \
                                        int prot); }
75                      STD             { int sys_madvise(void *addr, size_t len, \
                                int behav); }
76                      OBSOL      vhangup
77                      OBSOL      vlimit
78                      STD        { int sys_mincore(void *addr, size_t len, \
                                char *vec); }
79                      STD        { int sys_getgroups(int gidsetsize, \
                                gid_t *gidset); }
80                      STD        { int sys_setgroups(int gidsetsize, \
                                const gid_t *gidset); }
81                      STD        { int sys_getpgrp(void); }
82                      STD        { int sys_setpgid(pid_t pid, int pgid); }
83                      STD        { int sys_setitimer(int which, \
                                const struct itimerval *itv, \
                                struct itimerval *oitv); }
84                      COMPAT_43  { int sys_wait(void); } owait
85                      COMPAT_25  { int sys_swapon(const char *name); }
86                      STD        { int sys_getitimer(int which, \
                                struct itimerval *itv); }
87                      COMPAT_43  { int sys_gethostname(char *hostname,
u_int len); } \
                                ogethostname
88                      COMPAT_43  { int sys_sethostname(char *hostname,
u_int len); } \
                                osethostname
89                      COMPAT_43  { int sys_getdtablesize(void); }
ogetdtablesize
90                      STD        { int sys_dup2(int from, int to); }
91                      UNIMPL     getdopt
```

```
92                      STD             { int sys_fcntl(int fd, int cmd, ... void *arg); }
93                      STD             { int sys_select(int nd, fd_set *in, fd_set
*ou, \
                                    fd_set *ex, struct timeval *tv); }
94                      UNIMPL          setdopt
95                      STD             { int sys_fsync(int fd); }
96                      STD             { int sys_setpriority(int which, id_t who, int
prio); }
97                      STD             { int sys_socket(int domain, int type, int
protocol); }
98                      STD             { int sys_connect(int s, const struct
sockaddr *name, \
                                    socklen_t namelen); }
99                      COMPAT_43       { int sys_accept(int s, caddr_t name, \
                                    int *anamelen); } oaccept
100                     STD             { int sys_getpriority(int which, id_t who); }
101                     COMPAT_43       { int sys_send(int s, caddr_t buf, int len, \
                                    int flags); } osend
102                     COMPAT_43       { int sys_recv(int s, caddr_t buf, int len, \
                                    int flags); } orecv
103                     STD             { int sys_sigreturn(struct sigcontext
*sigcntxp); }
104                     STD             { int sys_bind(int s, const struct sockaddr
*name, \
                                    socklen_t namelen); }
105                     STD             { int sys_setsockopt(int s, int level, int
name, \
                                    const void *val, socklen_t valsize); }
106                     STD             { int sys_listen(int s, int backlog); }
```

```
107                     OBSOL        vtimes
108                     COMPAT_43    { int sys_sigvec(int signum, struct sigvec
*nsv, \
                              struct sigvec *osv); } osigvec
109                     COMPAT_43    { int sys_sigblock(int mask); } osigblock
110                     COMPAT_43    { int sys_sigsetmask(int mask); }
osigsetmask
111                     STD          { int sys_sigsuspend(int mask); }
112                     COMPAT_43    { int sys_sigstack(struct sigstack *nss, \
                              struct sigstack *oss); } osigstack
113                     COMPAT_43    { int sys_recvmsg(int s, struct omsghdr
*msg, \
                              int flags); } orecvmsg
114                     COMPAT_43    { int sys_sendmsg(int s, caddr_t msg, int
flags); } \
                              osendmsg
115                     OBSOL        vtrace
116                     STD          { int sys_gettimeofday(struct timeval *tp, \
                              struct timezone *tzp); }
117                     STD          { int sys_getrusage(int who, struct rusage
*rusage); }
118                     STD          { int sys_getsockopt(int s, int level, int
name, \
                              void *val, socklen_t *avalsize); }
119                     OBSOL        resuba
120                     STD          { ssize_t sys_readv(int fd, \
                              const struct iovec *iovp, int iovcnt); }
121                     STD          { ssize_t sys_writev(int fd, \
                              const struct iovec *iovp, int iovcnt); }
```

```
122                 STD             { int sys_settimeofday(const struct timeval
*tv, \
                                const struct timezone *tzp); }
123                 STD             { int sys_fchown(int fd, uid_t uid, gid_t gid);
}
124                 STD             { int sys_fchmod(int fd, int mode); }
125                 COMPAT_43       { int sys_recvfrom(int s, caddr_t buf, size_t
len, \
                                int flags, caddr_t from, int *fromlenaddr); } \
                                orecvfrom
126                 STD             { int sys_setreuid(uid_t ruid, uid_t euid); }
127                 STD             { int sys_setregid(gid_t rgid, gid_t egid); }
128                 STD             { int sys_rename(const char *from, const
char *to); }
129                 COMPAT_43       { int sys_truncate(const char *path, long
length); } \
                                otruncate
130                 COMPAT_43       { int sys_ftruncate(int fd, long length); }
oftruncate
131                 STD             { int sys_flock(int fd, int how); }
132                 STD             { int sys_mkfifo(const char *path, int
mode); }
133                 STD             { ssize_t sys_sendto(int s, const void *buf, \
                                size_t len, int flags, const struct sockaddr *to, \
                                socklen_t tolen); }
134                 STD             { int sys_shutdown(int s, int how); }
135                 STD             { int sys_socketpair(int domain, int type, \
                                int protocol, int *rsv); }
136                 STD             { int sys_mkdir(const char *path, int mode);
}
```

```
137                     STD             { int sys_rmdir(const char *path); }
138                     STD             { int sys_utimes(const char *path, \
                                    const struct timeval *tptr); }
139                     OBSOL           4.2 sigreturn
140                     STD             { int sys_adjtime(const struct timeval
*delta, \
                                    struct timeval *olddelta); }
141                     COMPAT_43       { int sys_getpeername(int fdes, caddr_t asa,
\
                                    socklen_t *alen); } ogetpeername
142                     COMPAT_43       { int32_t sys_gethostid(void); } ogethostid
143                     COMPAT_43       { int sys_sethostid(int32_t hostid); }
osethostid
144                     COMPAT_43       { int sys_getrlimit(int which, \
                                    struct ogetrlimit *rlp); } ogetrlimit
145                     COMPAT_43       { int sys_setrlimit(int which, \
                                    struct ogetrlimit *rlp); } osetrlimit
146                     COMPAT_43       { int sys_killpg(int pgid, int signum); } okillpg
147                     STD             { int sys_setsid(void); }
148                     STD             { int sys_quotactl(const char *path, int cmd,
\
                                    int uid, char *arg); }
149                     COMPAT_43       { int sys_quota(void); } oquota
150                     COMPAT_43       { int sys_getsockname(int fdec, caddr_t asa,
\
                                    int *alen); } ogetsockname


; Syscalls 151-180 inclusive are reserved for vendor-specific
; system calls.  (This includes various calls added for compatibity
; with other Unix variants.)

; Some of these calls are now supported by BSD...

151                         UNIMPL

152                         UNIMPL

153                         UNIMPL

154                         UNIMPL

#if defined(NFSCLIENT) || defined(NFSSERVER)

155                         STD             { int sys_nfssvc(int flag, void *argp); }

#else

155                         UNIMPL

#endif

156                         COMPAT_43       { int sys_getdirentries(int fd, char *buf, \

                                        int count, long *basep); } ogetdirentries

157                         COMPAT_25       { int sys_statfs(const char *path, \

                                        struct ostatfs *buf); } ostatfs

158                         COMPAT_25       { int sys_fstatfs(int fd, struct ostatfs *buf); }
\

                                        ostatfs

159                         UNIMPL

160                         UNIMPL

161                         STD             { int sys_getfh(const char *fname, fhandle_t
*fhp); }

162                         COMPAT_09       { int sys_getdomainname(char
*domainname, int len); } \

                                        ogetdomainname

163                         COMPAT_09       { int sys_setdomainname(char
*domainname, int len); } \

                                        osetdomainname
```

```
164                     COMPAT_09     { int sys_uname(struct outsname *name); }
ouname

165                     STD           { int sys_sysarch(int op, void *parms); }

166                     UNIMPL

167                     UNIMPL

168                     UNIMPL

#if defined(SYSVSEM) && !defined(__LP64__)

169                     COMPAT_10     { int sys_semsys(int which, int a2, int a3, int
a4, \

                               int a5); } osemsys

#else

169                     UNIMPL        1.0 semsys

#endif

#if defined(SYSVMSG) && !defined(__LP64__)

170                     COMPAT_10     { int sys_msgsys(int which, int a2, int a3, int
a4, \

                               int a5, int a6); } omsgsys

#else

170                     UNIMPL        1.0 msgsys

#endif

#if defined(SYSVSHM) && !defined(__LP64__)

171                     COMPAT_10     { int sys_shmsys(int which, int a2, int a3, int
a4); } \

                               oshmsys

#else

171                     UNIMPL        1.0 shmsys

#endif

172                     UNIMPL
```

```
173                 STD         { ssize_t sys_pread(int fd, void *buf, \
                            size_t nbyte, int pad, off_t offset); }
174                 STD         { ssize_t sys_pwrite(int fd, const void *buf, \
                            size_t nbyte, int pad, off_t offset); }
175                 UNIMPL      ntp_gettime
176                 UNIMPL      ntp_adjtime
177                 UNIMPL
178                 UNIMPL
179                 UNIMPL
180                 UNIMPL

; Syscalls 181-199 are used by/reserved for BSD
181                 STD         { int sys_setgid(gid_t gid); }
182                 STD         { int sys_setegid(gid_t egid); }
183                 STD         { int sys_seteuid(uid_t euid); }
#ifdef LFS
184                 STD         { int lfs_bmapv(fsid_t *fsidp, \
                             struct block_info *blkiov, int blkcnt); }
185                 STD         { int lfs_markv(fsid_t *fsidp, \
                             struct block_info *blkiov, int blkcnt); }
186                 STD         { int lfs_segclean(fsid_t *fsidp, u_long
segment); }
187                 STD         { int lfs_segwait(fsid_t *fsidp, struct timeval
*tv); }
#else
184                 UNIMPL
185                 UNIMPL
186                 UNIMPL
```

```
187                     UNIMPL
#endif
188                     STD          { int sys_stat(const char *path, struct stat
*ub); }
189                     STD          { int sys_fstat(int fd, struct stat *sb); }
190                     STD          { int sys_lstat(const char *path, struct stat
*ub); }
191                     STD          { long sys_pathconf(const char *path, int
name); }
192                     STD          { long sys_fpathconf(int fd, int name); }
193                     STD          { int sys_swapctl(int cmd, const void *arg,
int misc); }
194                     STD          { int sys_getrlimit(int which, \
                                  struct rlimit *rlp); }
195                     STD          { int sys_setrlimit(int which, \
                                  const struct rlimit *rlp); }
196                     STD          { int sys_getdirentries(int fd, char *buf, \
                                  int count, long *basep); }
197                     STD          { void *sys_mmap(void *addr, size_t len, int
prot, \
                                  int flags, int fd, long pad, off_t pos); }
198                     INDIR        { quad_t sys___syscall(quad_t num, ...); }
199                     STD          { off_t sys_lseek(int fd, int pad, off_t offset,
\
                                  int whence); }
200                     STD          { int sys_truncate(const char *path, int pad,
\
                                  off_t length); }
201                     STD          { int sys_ftruncate(int fd, int pad, off_t
length); }
```

```
202             STD         { int sys___sysctl(int *name, u_int namelen, \
                        void *old, size_t *oldlenp, void *new, \
                        size_t newlen); }
203             STD         { int sys_mlock(const void *addr, size_t len); }
204             STD         { int sys_munlock(const void *addr, size_t len); }
205             STD         { int sys_undelete(const char *path); }
206             STD         { int sys_futimes(int fd, \
                        const struct timeval *tptr); }
207             STD         { pid_t sys_getpgid(pid_t pid); }
208             STD         { int sys_xfspioctl(int operation, char *a_pathP, \
                        int a_opcode, struct ViceIoctl *a_paramsP, \
                        int a_followSymlinks); }
209             UNIMPL
;
; Syscalls 210-219 are reserved for dynamically loaded syscalls
;
#ifdef LKM
210             NODEF       { int sys_lkmnosys(void); }
211             NODEF       { int sys_lkmnosys(void); }
212             NODEF       { int sys_lkmnosys(void); }
213             NODEF       { int sys_lkmnosys(void); }
214             NODEF       { int sys_lkmnosys(void); }
215             NODEF       { int sys_lkmnosys(void); }
216             NODEF       { int sys_lkmnosys(void); }
```

```
217                       NODEF         { int sys_lkmnosys(void); }

218                       NODEF         { int sys_lkmnosys(void); }

219                       NODEF         { int sys_lkmnosys(void); }

#else                     /* !LKM */

210                       UNIMPL

211                       UNIMPL

212                       UNIMPL

213                       UNIMPL

214                       UNIMPL

215                       UNIMPL

216                       UNIMPL

217                       UNIMPL

218                       UNIMPL

219                       UNIMPL

#endif                    /* !LKM */

; System calls 220-240 are reserved for use by OpenBSD

#ifdef SYSVSEM

220                       COMPAT_23     { int sys___semctl(int semid, int semnum,
int cmd, \

                                    union semun *arg); } __osemctl

221                       STD           { int sys_semget(key_t key, int nsems, int
semflg); }

222                       STD           { int sys_semop(int semid, struct sembuf
*sops, \

                                    u_int nsops); }

223                       OBSOL         sys_semconfig

#else

220                       UNIMPL        semctl
```

```
221                 UNIMPL          semget
222                 UNIMPL          semop
223                 UNIMPL          semconfig
#endif
#ifdef SYSVMSG
224                 COMPAT_23       { int sys_msgctl(int msqid, int cmd, \
                                struct omsqid_ds *buf); } omsgctl
225                 STD             { int sys_msgget(key_t key, int msgflg); }
226                 STD             { int sys_msgsnd(int msqid, const void
*msgp, size_t msgsz, \
                                int msgflg); }
227                 STD             { int sys_msgrcv(int msqid, void *msgp,
size_t msgsz, \
                                long msgtyp, int msgflg); }
#else
224                 UNIMPL          msgctl
225                 UNIMPL          msgget
226                 UNIMPL          msgsnd
227                 UNIMPL          msgrcv
#endif
#ifdef SYSVSHM
228                 STD             { void *sys_shmat(int shmid, const void
*shmaddr, \
                                int shmflg); }
229                 COMPAT_23       { int sys_shmctl(int shmid, int cmd, \
                                struct oshmid_ds *buf); } oshmctl
230                 STD             { int sys_shmdt(const void *shmaddr); }
```

```
231                     STD         { int sys_shmget(key_t key, int size, int
shmflg); }
#else
228                     UNIMPL      shmat
229                     UNIMPL      shmctl
230                     UNIMPL      shmdt
231                     UNIMPL      shmget
#endif
232                     STD         { int sys_clock_gettime(clockid_t clock_id, \
                                 struct timespec *tp); }
233                     STD         { int sys_clock_settime(clockid_t clock_id, \
                                 const struct timespec *tp); }
234                     STD         { int sys_clock_getres(clockid_t clock_id, \
                                 struct timespec *tp); }
235                     UNIMPL      timer_create
236                     UNIMPL      timer_delete
237                     UNIMPL      timer_settime
238                     UNIMPL      timer_gettime
239                     UNIMPL      timer_getoverrun
;
; System calls 240-249 are reserved for other IEEE Std1003.1b syscalls
;
240                     STD         { int sys_nanosleep(const struct timespec
*rqtp, \
                                 struct timespec *rmtp); }
241                     UNIMPL
242                     UNIMPL
243                     UNIMPL
```

```
244         UNIMPL
245         UNIMPL
246         UNIMPL
247         UNIMPL
248         UNIMPL
249         UNIMPL
250         STD         { int sys_minherit(void *addr, size_t len, \
                    int inherit); }
251         STD         { int sys_rfork(int flags); }
252         STD         { int sys_poll(struct pollfd *fds, \
                    u_int nfds, int timeout); }
253         STD         { int sys_issetugid(void); }
254         STD         { int sys_lchown(const char *path, uid_t uid,
gid_t gid); }
255         STD         { pid_t sys_getsid(pid_t pid); }
256         STD         { int sys_msync(void *addr, size_t len, int
flags); }
#ifdef SYSVSEM
257         STD         { int sys___semctl(int semid, int semnum,
int cmd, \
                    union semun *arg); }
#else
257         UNIMPL
#endif
#ifdef SYSVSHM
258         STD         { int sys_shmctl(int shmid, int cmd, \
                    struct shmid_ds *buf); }
#else
```

```
258                     UNIMPL
#endif
#ifdef SYSVMSG
259                     STD             { int sys_msgctl(int msqid, int cmd, \
                                      struct msqid_ds *buf); }
#else
259                     UNIMPL
#endif
260                     STD             { int sys_getfsstat(struct statfs *buf, size_t
bufsize, \
                                      int flags); }
261                     STD             { int sys_statfs(const char *path, \
                                      struct statfs *buf); }
262                     STD             { int sys_fstatfs(int fd, struct statfs *buf); }
263                     STD             { int sys_pipe(int *fdp); }
264                     STD             { int sys_fhopen(const fhandle_t *fhp, int
flags); }
265                     STD             { int sys_fhstat(const fhandle_t *fhp, \
                                      struct stat *sb); }
266                     STD             { int sys_fhstatfs(const fhandle_t *fhp, \
                                      struct statfs *buf); }
267                     STD             { ssize_t sys_preadv(int fd, \
                                     const struct iovec *iovp, int iovcnt, \
                                     int pad, off_t offset); }
268                     STD             { ssize_t sys_pwritev(int fd, \
                                     const struct iovec *iovp, int iovcnt, \
                                     int pad, off_t offset); }
269                     STD             { int sys_kqueue(void); }
```

```
270                     STD             { int sys_kevent(int fd, \
                                    const struct kevent *changelist, int nchanges, \
                                    struct kevent *eventlist, int nevents, \
                                    const struct timespec *timeout); }
271                     STD             { int sys_mlockall(int flags); }
272                     STD             { int sys_munlockall(void); }
273                     STD             { int sys_getpeereid(int fdes, uid_t *euid,
gid_t *egid); }
#ifdef UFS_EXTATTR
274                     STD             { int sys_extattrctl(const char *path, int
cmd, \
                                    const char *filename, int attrnamespace, \
                                    const char *attrname); }
275                     STD             { int sys_extattr_set_file(const char *path, \
                                    int attrnamespace, const char *attrname, \
                                    void *data, size_t nbytes); }
276                     STD             { ssize_t sys_extattr_get_file(const char
*path, \
                                    int attrnamespace, const char *attrname, \
                                    void *data, size_t nbytes); }
277                     STD             { int sys_extattr_delete_file(const char
*path, \
                                    int attrnamespace, const char *attrname); }
278                     STD             { int sys_extattr_set_fd(int fd, int
attrnamespace, \
                                    const char *attrname, void *data, \
                                    size_t nbytes); }
279                     STD             { ssize_t sys_extattr_get_fd(int fd, \
                                    int attrnamespace, const char *attrname, \
                                        void *data, size_t nbytes); }
280                     STD                 { int sys_extattr_delete_fd(int fd, int
attrnamespace, \
                                        const char *attrname); }
#else
274                     UNIMPL              sys_extattrctl
275                     UNIMPL              sys_extattr_set_file
276                     UNIMPL              sys_extattr_get_file
277                     UNIMPL              sys_extattr_delete_file
278                     UNIMPL              sys_extattr_set_fd
279                     UNIMPL              sys_extattr_get_fd
280                     UNIMPL              sys_extattr_delete_fd
#endif
281                     STD                 { int sys_getresuid(uid_t *ruid, uid_t *euid,
\
                                        uid_t *suid); }
282                     STD                 { int sys_setresuid(uid_t ruid, uid_t euid, \
                                        uid_t suid); }
283                     STD                 { int sys_getresgid(gid_t *rgid, gid_t *egid, \
                                        gid_t *sgid); }
284                     STD                 { int sys_setresgid(gid_t rgid, gid_t egid, \
                                        gid_t sgid); }
285                     OBSOL               sys_omquery
286                     STD                 { void *sys_mquery(void *addr, size_t len,
int prot, \
                                        int flags, int fd, long pad, off_t pos); }
287                     STD                 { int sys_closefrom(int fd); }
```

```
288                                      STD                  { int sys_sigaltstack(const struct sigaltstack
*nss, \
                                                        struct sigaltstack *oss); }


;================================================================================
; COP4600 syscalls
;================================================================================


;added by Dave Small
289                                      STD                  { int sys_hello( void ); }
290                                      STD                  { int sys_showargs( const char *str, int val );
}
; added by Edward Tischler (Part 5 only)
291 STD                                  { int sys_allocate_semaphore( const char* name, int
initial_count ); }
292 STD                                  { int sys_down_semaphore( const char* name ); }
293 STD                                  { int sys_up_semaphore( const char* name );}
294 STD                                  { int sys_free_semaphore( const char* name );}
```

INIT_MAIN.C

USR/SRC/SYS/KERN/INIT_MAIN

```
/*					$OpenBSD: init_main.c,v 1.112 2004/03/14 23:12:11 tedu
Exp $					*/
/*					$NetBSD: init_main.c,v 1.84.4.1 1996/06/02 09:08:06 mrg
Exp $					*/


/*
 * Copyright (c) 1995 Christopher G. Demetriou.  All rights reserved.
 * Copyright (c) 1982, 1986, 1989, 1991, 1992, 1993
 *					The Regents of the University of California.  All rights
reserved.
 * (c) UNIX System Laboratories, Inc.
 * All or some portions of this file are derived from material licensed
 * to the University of California by American Telephone and Telegraph
 * Co. or Unix System Laboratories, Inc. and are reproduced herein with
 * the permission of UNIX System Laboratories, Inc.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *					@(#)init_main.c 8.9 (Berkeley) 1/21/94
 */
//#include <sys/malloc.h>
#include <sys/param.h>
#include <sys/filedesc.h>
#include <sys/file.h>
#include <sys/errno.h>
#include <sys/exec.h>
#include <sys/kernel.h>
#include <sys/kthread.h>
#include <sys/mount.h>
#include <sys/proc.h>
#include <sys/resourcevar.h>
#include <sys/signalvar.h>
#include <sys/systm.h>
```

```
#include <sys/namei.h>
#include <sys/vnode.h>
#include <sys/tty.h>
#include <sys/conf.h>
#include <sys/buf.h>
#include <sys/device.h>
#include <sys/socketvar.h>
#include <sys/lockf.h>
#include <sys/protosw.h>
#include <sys/reboot.h>
#include <sys/user.h>
#ifdef SYSVSHM
#include <sys/shm.h>
#endif
#ifdef SYSVSEM
#include <sys/sem.h>
#endif
#ifdef SYSVMSG
#include <sys/msg.h>
#endif
#include <sys/domain.h>
#include <sys/mbuf.h>
#include <sys/pipe.h>

#include <sys/syscall.h>
#include <sys/syscallargs.h>
```

```
#include <dev/rndvar.h>


#include <ufs/ufs/quota.h>


#include <machine/cpu.h>


#include <uvm/uvm.h>


#include <net/if.h>
#include <net/raw_cb.h>


/*****BEGIN ADDITION by Edward Tischler *********************/
 #include <sys/cop4600.h>
/*****END ADDITION by Edward Tischler ***********************/


#if defined(CRYPTO)
#include <crypto/cryptodev.h>
#include <crypto/cryptosoft.h>
#endif


#if defined(NFSSERVER) || defined(NFSCLIENT)
extern void nfs_init(void);
#endif


const char                              copyright[] =
"Copyright (c) 1982, 1986, 1989, 1991, 1993\n"
"\tThe Regents of the University of California.  All rights reserved.\n"
```

```
"Copyright (c) 1995-2004 OpenBSD. All rights reserved.  http://www.OpenBSD.org\n";

/* Components of the first process -- never freed. */
struct                  session session0;
struct                  pgrp pgrp0;
struct                  proc proc0;
struct                  pcred cred0;
struct                  plimit limit0;
struct                  vmspace vmspace0;
struct                  sigacts sigacts0;
#ifndef curproc
struct                  proc *curproc;
#endif
struct                  proc *initproc;

int                     cmask = CMASK;
extern                  struct user *proc0paddr;

void                    (*md_diskconf)(void) = NULL;
struct                  vnode *rootvp, *swapdev_vp;
int                     boothowto;
struct                  timeval boottime;
struct                  timeval runtime;

#if !defined(NO_PROPOLICE)
long                    __guard[8];
#endif
```

```
/* XXX return int so gcc -Werror won't complain */
int                                     main(void *);
void                                    check_console(struct proc *);
void                                    start_init(void *);
void                                    start_cleaner(void *);
void                                    start_update(void *);
void                                    start_reaper(void *);
void    start_crypto(void *);
void                                    init_exec(void);

extern char sigcode[], esigcode[];
#ifdef SYSCALL_DEBUG
extern char *syscallnames[];
#endif

struct emul emul_native = {
                                        "native",
                                        NULL,
                                        sendsig,
                                        SYS_syscall,
                                        SYS_MAXSYSCALL,
                                        sysent,
#ifdef SYSCALL_DEBUG
                                        syscallnames,
#else
                                        NULL,
#endif
                                        0,
                                        copyargs,
                                        setregs,
                                        NULL,
                                        sigcode,
                                        esigcode,
                                        EMUL_ENABLED | EMUL_NATIVE,
};


/*
 * System startup; initialize the world, create process 0, mount root
 * filesystem, and fork to create init and pagedaemon.  Most of the
 * hard work is done in the lower-level initialization routines including
 * startup(), which does memory initialization and autoconfiguration.
 */
/* XXX return int, so gcc -Werror won't complain */
int
main(framep)
                                        void *framep;                   /* XXX should go
away */
{
                                        struct proc *p;
                                        struct pdevinit *pdev;
                                        struct timeval rtv;
                                        quad_t lim;
```

```
int s, i;
register_t rval[2];
extern struct pdevinit pdevinit[];
extern void scheduler_start(void);
extern void disk_init(void);
extern void endtsleep(void *);
extern void realitexpire(void *);

/*
 * Initialize the current process pointer (curproc) before
 * any possible traps/probes to simplify trap processing.
 */
curproc = p = &proc0;

/*
 * Initialize timeouts.
 */
timeout_startup();

/*
 * Attempt to find console and initialize
 * in case of early panic or other messages.
 */
config_init();                  /* init autoconfiguration data
structures */
consinit();
printf("%s\n", copyright);
```

```
                        uvm_init();
                        disk_init();            /* must come before
autoconfiguration */
                        tty_init();             /* initialise tty's */
                        cpu_startup();

                        /*
                         * Initialize mbuf's.  Do this now because we might attempt
to
                         * allocate mbufs or mbuf clusters during autoconfiguration.
                         */
                        mbinit();

                        /* Initalize sockets. */
                        soinit();

                        /* Initialize sysctls (must be done before any processes run)
*/
                        sysctl_init();

                        /*
                         * Initialize process and pgrp structures.
                         */
                        procinit();

                        /* Initialize file locking. */
                        lf_init();
```

```
        /*
         * Initialize filedescriptors.
         */
        filedesc_init();

        /*
         * Initialize pipes.
         */
        pipe_init();

/***** BEGIN ADDITION by Edward Tischler ****************************/

         cop4600_sema_init();
         cop4600_pes_init();
/***** END ADDITION by Edward Tischler ******************************/

        /*
         * Create process 0 (the swapper).
         */
        LIST_INSERT_HEAD(&allproc, p, p_list);
        p->p_pgrp = &pgrp0;
        LIST_INSERT_HEAD(PIDHASH(0), p, p_hash);
        LIST_INSERT_HEAD(PGRPHASH(0), &pgrp0, pg_hash);
        LIST_INIT(&pgrp0.pg_members);
        LIST_INSERT_HEAD(&pgrp0.pg_members, p, p_pglist);
```

```
                pgrp0.pg_session = &session0;
                session0.s_count = 1;
                session0.s_leader = p;

                p->p_flag = P_INMEM | P_SYSTEM | P_NOCLDWAIT;
                p->p_stat = SRUN;
                p->p_nice = NZERO;
                p->p_emul = &emul_native;
/******ADDITION by Edward Tischler*********************************/
                cop4600_pes_create(&p->pes, p);
/***** END ADDITION by Edward Tischler ***************************/


                bcopy("swapper", p->p_comm, sizeof ("swapper"));

                /* Init timeouts. */
                timeout_set(&p->p_sleep_to, endtsleep, p);
                timeout_set(&p->p_realit_to, realitexpire, p);

                /* Create credentials. */
                cred0.p_refcnt = 1;
                p->p_cred = &cred0;
                p->p_ucred = crget();
                p->p_ucred->cr_ngroups = 1;    /* group 0 */

                /* Initialize signal state for process 0. */
                signal_init();
```

```
                                p->p_sigacts = &sigacts0;
                                siginit(p);

                                /* Create the file descriptor table. */
                                p->p_fd = fdinit(NULL);

                                /* Create the limits structures. */
                                p->p_limit = &limit0;
                                for (i = 0; i < sizeof(p->p_rlimit)/sizeof(p->p_rlimit[0]); i++)
                                   limit0.pl_rlimit[i].rlim_cur =
                                       limit0.pl_rlimit[i].rlim_max = RLIM_INFINITY;
                                limit0.pl_rlimit[RLIMIT_NOFILE].rlim_cur = NOFILE;
                                limit0.pl_rlimit[RLIMIT_NOFILE].rlim_max =
MIN(NOFILE_MAX,
                                   (maxfiles - NOFILE > NOFILE) ?  maxfiles - NOFILE :
NOFILE);
                                limit0.pl_rlimit[RLIMIT_NPROC].rlim_cur = MAXUPRC;
                                lim = ptoa(uvmexp.free);
                                limit0.pl_rlimit[RLIMIT_RSS].rlim_max = lim;
                                limit0.pl_rlimit[RLIMIT_MEMLOCK].rlim_max = lim;
                                limit0.pl_rlimit[RLIMIT_MEMLOCK].rlim_cur = lim / 3;
                                limit0.p_refcnt = 1;

                                /* Allocate a prototype map so we have something to fork.
*/
                                uvmspace_init(&vmspace0, pmap_kernel(),
round_page(VM_MIN_ADDRESS),
                                   trunc_page(VM_MAX_ADDRESS), TRUE);
```

```
				p->p_vmspace = &vmspace0;

				p->p_addr = proc0paddr;				/* XXX
*/

				/*
				 * We continue to place resource usage info in the
				 * user struct so they're pageable.
				 */
				p->p_stats = &p->p_addr->u_stats;

				/*
				 * Charge root for one process.
				 */
				(void)chgproccnt(0, 1);

				/* Initialize run queues */
				rqinit();

				/* Configure the devices */
				cpu_configure();

				/* Configure virtual memory system, set vm rlimits. */
				uvm_init_limits(p);

				/* Initialize the file systems. */
#if defined(NFSSERVER) || defined(NFSCLIENT)
```

```
                                        nfs_init();                         /* initialize server/shared
data */
#endif
                                        vfsinit();

                                        /* Start real time and statistics clocks. */
                                        initclocks();

#ifdef SYSVSHM
                                        /* Initialize System V style shared memory. */
                                        shminit();
#endif

#ifdef SYSVSEM
                                        /* Initialize System V style semaphores. */
                                        seminit();
#endif

#ifdef SYSVMSG
                                        /* Initialize System V style message queues. */
                                        msginit();
#endif

                                        /* Attach pseudo-devices. */
                                        randomattach();
                                        for (pdev = pdevinit; pdev->pdev_attach != NULL; pdev++)
                                           if (pdev->pdev_count > 0)
```

```
                (*pdev->pdev_attach)(pdev->pdev_count);

#ifdef CRYPTO
        swcr_init();
#endif /* CRYPTO */

        /*
         * Initialize protocols.  Block reception of incoming packets
         * until everything is ready.
         */
        s = splimp();
        ifinit();
        domaininit();
        if_attachdomain();
        splx(s);

#ifdef GPROF
        /* Initialize kernel profiling. */
        kmstartup();
#endif

#if !defined(NO_PROPOLICE)
        arc4random_bytes(__guard, sizeof(__guard));
#endif

        /* init exec and emul */
        init_exec();
```

```
/* Start the scheduler */
scheduler_start();

dostartuphooks();

/* Configure root/swap devices */
if (md_diskconf)
   (*md_diskconf)();

/* Mount the root file system. */
if (vfs_mountroot())
   panic("cannot mount root");
CIRCLEQ_FIRST(&mountlist)->mnt_flag |= MNT_ROOTFS;

/* Get the vnode for '/'.  Set p->p_fd->fd_cdir to reference it.
*/

if (VFS_ROOT(mountlist.cqh_first, &rootvnode))
   panic("cannot find root vnode");
p->p_fd->fd_cdir = rootvnode;
VREF(p->p_fd->fd_cdir);
VOP_UNLOCK(rootvnode, 0, p);
p->p_fd->fd_rdir = NULL;

uvm_swap_init();

/*
```

```
                 * Now can look at time, having had a chance to verify the
time
                 * from the file system.  Reset p->p_rtime as it may have
been
                 * munched in mi_switch() after the time got set.
                 */
                p->p_stats->p_start = runtime = mono_time = boottime =
time;
                p->p_rtime.tv_sec = p->p_rtime.tv_usec = 0;

                /* Create process 1 (init(8)). */
                if (fork1(p, SIGCHLD, FORK_FORK, NULL, 0, start_init, NULL,
rval))
                   panic("fork init");

                /* Create process 2, the pageout daemon kernel thread. */
                if (kthread_create(uvm_pageout, NULL, NULL,
"pagedaemon"))
                   panic("fork pagedaemon");

                /* Create process 3, the reaper daemon kernel thread. */
                if (kthread_create(start_reaper, NULL, NULL, "reaper"))
                   panic("fork reaper");

                /* Create process 4, the cleaner daemon kernel thread. */
                if (kthread_create(start_cleaner, NULL, NULL, "cleaner"))
                   panic("fork cleaner");

                /* Create process 5, the update daemon kernel thread. */
```

```
                                        if (kthread_create(start_update, NULL, NULL, "update"))
                                          panic("fork update");

                                        /* Create process 6, the aiodone daemon kernel thread. */
                                        if (kthread_create(uvm_aiodone_daemon, NULL, NULL,
"aiodoned"))
                                          panic("fork aiodoned");

#ifdef CRYPTO
                                        /* Create process 7, the crypto kernel thread. */
                                        if (kthread_create(start_crypto, NULL, NULL, "crypto"))
                                          panic("crypto thread");
#endif /* CRYPTO */

                                        /* Create any other deferred kernel threads. */
                                        kthread_run_deferred_queue();

                                        microtime(&rtv);
                                        srandom((u_long)(rtv.tv_sec ^ rtv.tv_usec));

                                        randompid = 1;
                                        /* The scheduler is an infinite loop. */
                                        uvm_scheduler();
                                        /* NOTREACHED */
}

/*
```

```
 * List of paths to try when searching for "init".
 */
static char *initpaths[] = {
                        "/sbin/init",
                        "/sbin/oinit",
                        "/sbin/init.bak",
                        NULL,
};

void
check_console(p)
                        struct proc *p;
{
                        struct nameidata nd;
                        int error;

                        NDINIT(&nd, LOOKUP, FOLLOW, UIO_SYSSPACE,
"/dev/console", p);
                        error = namei(&nd);
                        if (error) {
                          if (error == ENOENT)
                                printf("warning: /dev/console does not exist\n");
                          else
                                printf("warning: /dev/console error %d\n", error);
                        } else
                          vrele(nd.ni_vp);
}
```

```
/*
 * Start the initial user process; try exec'ing each pathname in "initpaths".
 * The program is invoked with one argument containing the boot flags.
 */
void
start_init(arg)
                                   void *arg;
{
                                   struct proc *p = arg;
                                   vaddr_t addr;
                                   struct sys_execve_args /* {
                                      syscallarg(const char *) path;
                                      syscallarg(char *const *) argp;
                                      syscallarg(char *const *) envp;
                                   } */ args;
                                   int options, error;
                                   long i;
                                   register_t retval[2];
                                   char flags[4], *flagsp;
                                   char **pathp, *path, *ucp, **uap, *arg0, *arg1 = NULL;

                                   initproc = p;

                                   /*
                                    * Now in process 1.
                                    */
```

```
                                        check_console(p);


                                        /*
                                         * Need just enough stack to hold the faked-up "execve()"
arguments.
                                         */
#ifdef MACHINE_STACK_GROWS_UP
                                        addr = USRSTACK;
#else
                                        addr = USRSTACK - PAGE_SIZE;
#endif
                                        if (uvm_map(&p->p_vmspace->vm_map, &addr, PAGE_SIZE,
                                          NULL, UVM_UNKNOWN_OFFSET, 0,
                                          UVM_MAPFLAG(UVM_PROT_RW, UVM_PROT_ALL,
UVM_INH_COPY,
                                          UVM_ADV_NORMAL,
UVM_FLAG_FIXED|UVM_FLAG_OVERLAY|UVM_FLAG_COPYONW)))
                                          panic("init: couldn't allocate argument space");
                                        p->p_vmspace->vm_maxsaddr = (caddr_t)addr;


                                        for (pathp = &initpaths[0]; (path = *pathp) != NULL;
pathp++) {
#ifdef MACHINE_STACK_GROWS_UP
                                          ucp = (char *)addr;
#else
                                          ucp = (char *)(addr + PAGE_SIZE);
#endif
                                          /*
```

```
                 * Construct the boot flag argument.
                 */
                flagsp = flags;
                *flagsp++ = '-';
                options = 0;

                if (boothowto & RB_SINGLE) {
                        *flagsp++ = 's';
                        options = 1;
                }
#ifdef notyet
                if (boothowto & RB_FASTBOOT) {
                        *flagsp++ = 'f';
                        options = 1;
                }
#endif

                /*
                 * Move out the flags (arg 1), if necessary.
                 */
                if (options != 0) {
                        *flagsp++ = '\0';
                        i = flagsp - flags;
#ifdef DEBUG
                        printf("init: copying out flags `%s' %d\n", flags, i);
#endif
#ifdef MACHINE_STACK_GROWS_UP
```

```
                                                arg1 = ucp;
                                                (void)copyout((caddr_t)flags, (caddr_t)ucp, i);
                                                ucp += i;
#else
                                                (void)copyout((caddr_t)flags, (caddr_t)(ucp -= i), i);
                                                arg1 = ucp;
#endif
                                        }

                                        /*
                                         * Move out the file name (also arg 0).
                                         */
                                        i = strlen(path) + 1;
#ifdef DEBUG
                                        printf("init: copying out path `%s' %d\n", path, i);
#endif
#ifdef MACHINE_STACK_GROWS_UP
                                        arg0 = ucp;
                                        (void)copyout((caddr_t)path, (caddr_t)ucp, i);
                                        ucp += i;
                                        ucp = (caddr_t)ALIGN((u_long)ucp);
                                        uap = (char **)ucp + 3;
#else
                                        (void)copyout((caddr_t)path, (caddr_t)(ucp -= i), i);
                                        arg0 = ucp;
                                        uap = (char **)((u_long)ucp & ~ALIGNBYTES);
#endif
```

```
    /*
     * Move out the arg pointers.
     */
    i = 0;
    copyout(&i, (caddr_t)--uap, sizeof(register_t)); /*
terminator */
    if (options != 0)
            copyout(&arg1, (caddr_t)--uap, sizeof(register_t));
    copyout(&arg0, (caddr_t)--uap, sizeof(register_t));

    /*
     * Point at the arguments.
     */
    SCARG(&args, path) = arg0;
    SCARG(&args, argp) = uap;
    SCARG(&args, envp) = NULL;

    /*
     * Now try to exec the program.  If can't for any reason
     * other than it doesn't exist, complain.
     */
    if ((error = sys_execve(p, &args, retval)) == 0)
            return;
    if (error != ENOENT)
            printf("exec %s: error %d\n", path, error);
}
```

```
		printf("init: not found\n");
		panic("no init");
}

void
start_update(arg)
		void *arg;
{
		sched_sync(curproc);
		/* NOTREACHED */
}

void
start_cleaner(arg)
		void *arg;
{
		buf_daemon(curproc);
		/* NOTREACHED */
}

void
start_reaper(arg)
		void *arg;
{
		reaper();
		/* NOTREACHED */
}
```

```
#ifdef CRYPTO
void
start_crypto(arg)
                                void *arg;
{
                                crypto_thread();
                                /* NOTREACHED */
}
#endif /* CRYPTO */
```

KERN_FORK.C

USR/SRC/SYS/KERN/KERN_FORK.C

```
/*                                      $OpenBSD: kern_fork.c,v 1.63 2003/09/23 20:26:18 millert
Exp $                                   */

/*                                      $NetBSD: kern_fork.c,v 1.29 1996/02/09 18:59:34 christos
Exp $                                   */


/*

 * Copyright (c) 1982, 1986, 1989, 1991, 1993

 *                                      The Regents of the University of California.  All rights
reserved.

 * (c) UNIX System Laboratories, Inc.

 * All or some portions of this file are derived from material licensed

 * to the University of California by American Telephone and Telegraph

 * Co. or Unix System Laboratories, Inc. and are reproduced herein with

 * the permission of UNIX System Laboratories, Inc.

 *

 * Redistribution and use in source and binary forms, with or without

 * modification, are permitted provided that the following conditions

 * are met:

 * 1. Redistributions of source code must retain the above copyright

 *    notice, this list of conditions and the following disclaimer.

 * 2. Redistributions in binary form must reproduce the above copyright

 *    notice, this list of conditions and the following disclaimer in the

 *    documentation and/or other materials provided with the distribution.

 * 3. Neither the name of the University nor the names of its contributors

 *    may be used to endorse or promote products derived from this software

 *    without specific prior written permission.

 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *                              @(#)kern_fork.c 8.6 (Berkeley) 4/8/94
 */

#include <sys/param.h>
#include <sys/systm.h>
#include <sys/filedesc.h>
#include <sys/kernel.h>
#include <sys/malloc.h>
#include <sys/mount.h>
#include <sys/proc.h>
#include <sys/resourcevar.h>
#include <sys/signalvar.h>
#include <sys/vnode.h>
#include <sys/file.h>
#include <sys/acct.h>
```

```
#include <sys/ktrace.h>
#include <sys/sched.h>
#include <dev/rndvar.h>
#include <sys/pool.h>
#include <sys/mman.h>

#include <sys/syscallargs.h>

#include "systrace.h"
#include <dev/systrace.h>

#include <uvm/uvm_extern.h>
#include <uvm/uvm_map.h>

/*****BEGIN ADDITION by Edward Tischler *********************/
 #include <sys/cop4600.h>
/*****END ADDITION by Edward Tischler ***********************/

int                     nprocs = 1;             /* process 0 */
int                     randompid;              /* when set to 1, pid's go random */
pid_t                   lastpid;
struct                  forkstat forkstat;

int pidtaken(pid_t);

/*ARGSUSED*/
int
```

```
sys_fork(struct proc *p, void *v, register_t *retval)
{
        return (fork1(p, SIGCHLD, FORK_FORK, NULL, 0, NULL, NULL,
retval));
}

/*ARGSUSED*/
int
sys_vfork(struct proc *p, void *v, register_t *retval)
{
        return (fork1(p, SIGCHLD, FORK_VFORK|FORK_PPWAIT,
NULL, 0, NULL,
            NULL, retval));
}

int
sys_rfork(struct proc *p, void *v, register_t *retval)
{
        struct sys_rfork_args /* {
           syscallarg(int) flags;
        } */ *uap = v;

        int rforkflags;
        int flags;

        flags = FORK_RFORK;
        rforkflags = SCARG(uap, flags);
        if ((rforkflags & RFPROC) == 0)
           return (EINVAL);

        switch(rforkflags & (RFFDG|RFCFDG)) {
        case (RFFDG|RFCFDG):
           return EINVAL;
        case RFCFDG:
           flags |= FORK_CLEANFILES;
           break;
        case RFFDG:
           break;
        default:
           flags |= FORK_SHAREFILES;
           break;
        }

        if (rforkflags & RFNOWAIT)
           flags |= FORK_NOZOMBIE;

        if (rforkflags & RFMEM)
           flags |= FORK_VMNOSTACK;

        return (fork1(p, SIGCHLD, flags, NULL, 0, NULL, NULL,
retval));
}

/* print the 'table full' message once per 10 seconds */
```

```
struct timeval fork_tfmrate = { 10, 0 };

int
fork1(struct proc *p1, int exitsig, int flags, void *stack, size_t stacksize,
    void (*func)(void *), void *arg, register_t *retval)
{
        struct proc *p2;
        uid_t uid;
        struct vmspace *vm;
        int count;
        vaddr_t uaddr;
        int s;
        extern void endtsleep(void *);
        extern void realitexpire(void *);

        /*
         * Although process entries are dynamically created, we still keep
         * a global limit on the maximum number we will create. We reserve
         * the last 5 processes to root. The variable nprocs is the current
         * number of processes, maxproc is the limit.
         */
        uid = p1->p_cred->p_ruid;
        if ((nprocs >= maxproc - 5 && uid != 0) || nprocs >= maxproc) {
            static struct timeval lasttfm;
```

```
   if (ratecheck(&lasttfm, &fork_tfmrate))
           tablefull("proc");
   return (EAGAIN);
}
nprocs++;

/*
 * Increment the count of procs running with this uid. Don't
allow
 * a nonprivileged user to exceed their current limit.
 */
count = chgproccnt(uid, 1);
if (uid != 0 && count > p1-
>p_rlimit[RLIMIT_NPROC].rlim_cur) {
   (void)chgproccnt(uid, -1);
   nprocs--;
   return (EAGAIN);
}

/*
 * Allocate a pcb and kernel stack for the process
 */
uaddr = uvm_km_valloc(kernel_map, USPACE);
if (uaddr == 0) {
   chgproccnt(uid, -1);
   nprocs--;
   return (ENOMEM);
```

```
                        }

                        /*
                         * From now on, we're committed to the fork and cannot
fail.
                         */

                        /* Allocate new proc. */
                        p2 = pool_get(&proc_pool, PR_WAITOK);

/******ADDITION by Edward Tischler**********************************/
                        cop4600_pes_create(&p2->pes , p2);
/***** END ADDITION by Edward Tischler ****************************/

                        p2->p_stat = SIDL;                    /* protect against
others */
                        p2->p_exitsig = exitsig;
                        p2->p_forw = p2->p_back = NULL;

                        /*
                         * Make a proc table entry for the new process.
                         * Start by zeroing the section of proc that is zero-initialized,
                         * then copy the section that is copied directly from the
parent.
                         */
                        bzero(&p2->p_startzero,
                            (unsigned) ((caddr_t)&p2->p_endzero - (caddr_t)&p2-
>p_startzero));
```

```
bcopy(&p1->p_startcopy, &p2->p_startcopy,
   (unsigned) ((caddr_t)&p2->p_endcopy - (caddr_t)&p2-
>p_startcopy));

/*
 * Initialize the timeouts.
 */
timeout_set(&p2->p_sleep_to, endtsleep, p2);
timeout_set(&p2->p_realit_to, realitexpire, p2);

/*
 * Duplicate sub-structures as needed.
 * Increase reference counts on shared objects.
 * The p_stats and p_sigacts substructs are set in vm_fork.
 */
p2->p_flag = P_INMEM;
p2->p_emul = p1->p_emul;

if (p1->p_flag & P_PROFIL)
   startprofclock(p2);
p2->p_flag |= (p1->p_flag & (P_SUGID | P_SUGIDEXEC));
p2->p_cred = pool_get(&pcred_pool, PR_WAITOK);
bcopy(p1->p_cred, p2->p_cred, sizeof(*p2->p_cred));
p2->p_cred->p_refcnt = 1;
crhold(p1->p_ucred);

/* bump references to the text vnode (for procfs) */
```

```
p2->p_textvp = p1->p_textvp;
if (p2->p_textvp)
   VREF(p2->p_textvp);

if (flags & FORK_CLEANFILES)
   p2->p_fd = fdinit(p1);
else if (flags & FORK_SHAREFILES)
   p2->p_fd = fdshare(p1);
else
   p2->p_fd = fdcopy(p1);

/*
 * If p_limit is still copy-on-write, bump refcnt,
 * otherwise get a copy that won't be modified.
 * (If PL_SHAREMOD is clear, the structure is shared
 * copy-on-write.)
 */
if (p1->p_limit->p_lflags & PL_SHAREMOD)
   p2->p_limit = limcopy(p1->p_limit);
else {
   p2->p_limit = p1->p_limit;
   p2->p_limit->p_refcnt++;
}

if (p1->p_session->s_ttyvp != NULL && p1->p_flag &
P_CONTROLT)
   p2->p_flag |= P_CONTROLT;
```

```
                    if (flags & FORK_PPWAIT)
                       p2->p_flag |= P_PPWAIT;
                    LIST_INSERT_AFTER(p1, p2, p_pglist);
                    p2->p_pptr = p1;
                    if (flags & FORK_NOZOMBIE)
                       p2->p_flag |= P_NOZOMBIE;
                    LIST_INSERT_HEAD(&p1->p_children, p2, p_sibling);
                    LIST_INIT(&p2->p_children);

#ifdef KTRACE
                    /*
                     * Copy traceflag and tracefile if enabled.
                     * If not inherited, these were zeroed above.
                     */
                    if (p1->p_traceflag & KTRFAC_INHERIT) {
                       p2->p_traceflag = p1->p_traceflag;
                       if ((p2->p_tracep = p1->p_tracep) != NULL)
                                 VREF(p2->p_tracep);
                    }
#endif

                    /*
                     * set priority of child to be that of parent
                     * XXX should move p_estcpu into the region of struct proc
which gets
                     * copied.
                     */
```

```
scheduler_fork_hook(p1, p2);


/*
 * Create signal actions for the child process.
 */
if (flags & FORK_SIGHAND)
   sigactsshare(p1, p2);
else
   p2->p_sigacts = sigactsinit(p1);


/*
 * If emulation has process fork hook, call it now.
 */
if (p2->p_emul->e_proc_fork)
   (*p2->p_emul->e_proc_fork)(p2, p1);
/*
 * This begins the section where we must prevent the parent
 * from being swapped.
 */
PHOLD(p1);


if (flags & FORK_VMNOSTACK) {
   /* share everything, but ... */
   uvm_map_inherit(&p1->p_vmspace->vm_map,
       VM_MIN_ADDRESS, VM_MAXUSER_ADDRESS,
       MAP_INHERIT_SHARE);
   /* ... don't share stack */
```

```
#ifdef MACHINE_STACK_GROWS_UP
                                        uvm_map_inherit(&p1->p_vmspace->vm_map,
                                            USRSTACK, USRSTACK + MAXSSIZ,
                                            MAP_INHERIT_COPY);
#else
                                        uvm_map_inherit(&p1->p_vmspace->vm_map,
                                            USRSTACK - MAXSSIZ, USRSTACK,
                                            MAP_INHERIT_COPY);
#endif
                                    }

                                    p2->p_addr = (struct user *)uaddr;

                                    /*
                                     * Finish creating the child process.  It will return through a
                                     * different path later.
                                     */
                                    uvm_fork(p1, p2, ((flags & FORK_SHAREVM) ? TRUE : FALSE),
stack,
                                        stacksize, func ? func : child_return, arg ? arg : p2);

                                    vm = p2->p_vmspace;

                                    if (flags & FORK_FORK) {
                                        forkstat.cntfork++;
                                        forkstat.sizfork += vm->vm_dsize + vm->vm_ssize;
                                    } else if (flags & FORK_VFORK) {
```

```
                  forkstat.cntvfork++;
                  forkstat.sizvfork += vm->vm_dsize + vm->vm_ssize;
                } else if (flags & FORK_RFORK) {
                  forkstat.cntrfork++;
                  forkstat.sizrfork += vm->vm_dsize + vm->vm_ssize;
                } else {
                  forkstat.cntkthread++;
                  forkstat.sizkthread += vm->vm_dsize + vm->vm_ssize;
                }

                /* Find an unused pid satisfying 1 <= lastpid <= PID_MAX */
                do {
                  lastpid = 1 + (randompid ? arc4random() : lastpid) %
PID_MAX;
                } while (pidtaken(lastpid));
                p2->p_pid = lastpid;

                LIST_INSERT_HEAD(&allproc, p2, p_list);
                LIST_INSERT_HEAD(PIDHASH(p2->p_pid), p2, p_hash);

#if NSYSTRACE > 0
                if (ISSET(p1->p_flag, P_SYSTRACE))
                  systrace_fork(p1, p2);
#endif

                /*
                 * Make child runnable, set start time, and add to run queue.
```

```
 */
s = splstatclock();
p2->p_stats->p_start = time;
p2->p_acflag = AFORK;
p2->p_stat = SRUN;
setrunqueue(p2);
splx(s);

/*
 * Now can be swapped.
 */
PRELE(p1);

uvmexp.forks++;
if (flags & FORK_PPWAIT)
   uvmexp.forks_ppwait++;
if (flags & FORK_SHAREVM)
   uvmexp.forks_sharevm++;

/*
 * tell any interested parties about the new process
 */
KNOTE(&p1->p_klist, NOTE_FORK | p2->p_pid);

/*
 * Preserve synchronization semantics of vfork.  If waiting for
```

```
                         * child to exec or exit, set P_PPWAIT on child, and sleep on
our
                         * proc (in case of exit).
                         */
                        if (flags & FORK_PPWAIT)
                          while (p2->p_flag & P_PPWAIT)
                                  tsleep(p1, PWAIT, "ppwait", 0);

                        /*
                         * Return child pid to parent process,
                         * marking us as parent via retval[1].
                         */
                        retval[0] = p2->p_pid;
                        retval[1] = 0;
                        return (0);
}

/*
 * Checks for current use of a pid, either as a pid or pgid.
 */
int
pidtaken(pid_t pid)
{
                        struct proc *p;

                        if (pfind(pid) != NULL)
                          return (1);
```

```
                                        if (pgfind(pid) != NULL)
                                          return (1);
                                        LIST_FOREACH(p, &zombproc, p_list)
                                          if (p->p_pid == pid || p->p_pgid == pid)
                                                  return (1);
                                        return (0);
}
```

KERN_EXIT.C

USR/SRC/SYS/KERN/KERN_EXIT.C

```
/*                                  $OpenBSD: kern_fork.c,v 1.63 2003/09/23 20:26:18 millert
Exp $                               */

/*                                  $NetBSD: kern_fork.c,v 1.29 1996/02/09 18:59:34 christos
Exp $                               */


/*

 * Copyright (c) 1982, 1986, 1989, 1991, 1993

 *                                  The Regents of the University of California.  All rights
reserved.

 * (c) UNIX System Laboratories, Inc.

 * All or some portions of this file are derived from material licensed

 * to the University of California by American Telephone and Telegraph

 * Co. or Unix System Laboratories, Inc. and are reproduced herein with

 * the permission of UNIX System Laboratories, Inc.

 *

 * Redistribution and use in source and binary forms, with or without

 * modification, are permitted provided that the following conditions

 * are met:

 * 1. Redistributions of source code must retain the above copyright

 *    notice, this list of conditions and the following disclaimer.

 * 2. Redistributions in binary form must reproduce the above copyright

 *    notice, this list of conditions and the following disclaimer in the

 *    documentation and/or other materials provided with the distribution.

 * 3. Neither the name of the University nor the names of its contributors

 *    may be used to endorse or promote products derived from this software

 *    without specific prior written permission.

 *

 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *				@(#)kern_fork.c 8.6 (Berkeley) 4/8/94
 */

#include <sys/param.h>
#include <sys/systm.h>
#include <sys/filedesc.h>
#include <sys/kernel.h>
#include <sys/malloc.h>
#include <sys/mount.h>
#include <sys/proc.h>
#include <sys/resourcevar.h>
#include <sys/signalvar.h>
#include <sys/vnode.h>
#include <sys/file.h>
#include <sys/acct.h>
#include <sys/ktrace.h>
```

```
#include <sys/sched.h>
#include <dev/rndvar.h>
#include <sys/pool.h>
#include <sys/mman.h>

#include <sys/syscallargs.h>

#include "systrace.h"
#include <dev/systrace.h>

#include <uvm/uvm_extern.h>
#include <uvm/uvm_map.h>

/*****BEGIN ADDITION by Edward Tischler *********************/
 #include <sys/cop4600.h>
/*****END ADDITION by Edward Tischler ***********************/

int                     nprocs = 1;         /* process 0 */
int                     randompid;          /* when set to 1, pid's go random */
pid_t                   lastpid;
struct                  forkstat forkstat;

int pidtaken(pid_t);

/*ARGSUSED*/
int
sys_fork(struct proc *p, void *v, register_t *retval)
```

```
{
                        return (fork1(p, SIGCHLD, FORK_FORK, NULL, 0, NULL, NULL,
retval));
}

/*ARGSUSED*/
int
sys_vfork(struct proc *p, void *v, register_t *retval)
{
                        return (fork1(p, SIGCHLD, FORK_VFORK|FORK_PPWAIT,
NULL, 0, NULL,
                          NULL, retval));
}

int
sys_rfork(struct proc *p, void *v, register_t *retval)
{
                        struct sys_rfork_args /* {
                          syscallarg(int) flags;
                        } */ *uap = v;

                        int rforkflags;
                        int flags;

                        flags = FORK_RFORK;
                        rforkflags = SCARG(uap, flags);

                        if ((rforkflags & RFPROC) == 0)
```

```
                                    return (EINVAL);

                                  switch(rforkflags & (RFFDG|RFCFDG)) {
                                  case (RFFDG|RFCFDG):
                                    return EINVAL;
                                  case RFCFDG:
                                    flags |= FORK_CLEANFILES;
                                    break;
                                  case RFFDG:
                                    break;
                                  default:
                                    flags |= FORK_SHAREFILES;
                                    break;
                                  }

                                  if (rforkflags & RFNOWAIT)
                                    flags |= FORK_NOZOMBIE;

                                  if (rforkflags & RFMEM)
                                    flags |= FORK_VMNOSTACK;

                                  return (fork1(p, SIGCHLD, flags, NULL, 0, NULL, NULL,
retval));
}

/* print the 'table full' message once per 10 seconds */
struct timeval fork_tfmrate = { 10, 0 };
```

```
int
fork1(struct proc *p1, int exitsig, int flags, void *stack, size_t stacksize,
  void (*func)(void *), void *arg, register_t *retval)
{
                                    struct proc *p2;
                                    uid_t uid;
                                    struct vmspace *vm;
                                    int count;
                                    vaddr_t uaddr;
                                    int s;
                                    extern void endtsleep(void *);
                                    extern void realitexpire(void *);

                                    /*
                                     * Although process entries are dynamically created, we still
keep
                                     * a global limit on the maximum number we will create. We
reserve
                                     * the last 5 processes to root. The variable nprocs is the
current
                                     * number of processes, maxproc is the limit.
                                     */
                                    uid = p1->p_cred->p_ruid;
                                    if ((nprocs >= maxproc - 5 && uid != 0) || nprocs >=
maxproc) {
                                      static struct timeval lasttfm;
```

```
   if (ratecheck(&lasttfm, &fork_tfmrate))
            tablefull("proc");
   return (EAGAIN);
}
nprocs++;

/*
 * Increment the count of procs running with this uid. Don't allow
 * a nonprivileged user to exceed their current limit.
 */
count = chgproccnt(uid, 1);
if (uid != 0 && count > p1->p_rlimit[RLIMIT_NPROC].rlim_cur) {
   (void)chgproccnt(uid, -1);
   nprocs--;
   return (EAGAIN);
}

/*
 * Allocate a pcb and kernel stack for the process
 */
uaddr = uvm_km_valloc(kernel_map, USPACE);
if (uaddr == 0) {
   chgproccnt(uid, -1);
   nprocs--;
   return (ENOMEM);
}
```

```
	/*
	 * From now on, we're committed to the fork and cannot fail.
	 */

	/* Allocate new proc. */
	p2 = pool_get(&proc_pool, PR_WAITOK);

/******ADDITION by Edward Tischler*********************************/
	cop4600_pes_create(&p2->pes , p2);
/***** END ADDITION by Edward Tischler **************************/

	p2->p_stat = SIDL;			/* protect against others */
	p2->p_exitsig = exitsig;
	p2->p_forw = p2->p_back = NULL;

	/*
	 * Make a proc table entry for the new process.
	 * Start by zeroing the section of proc that is zero-initialized,
	 * then copy the section that is copied directly from the parent.
	 */
	bzero(&p2->p_startzero,
	    (unsigned) ((caddr_t)&p2->p_endzero - (caddr_t)&p2->p_startzero));
	bcopy(&p1->p_startcopy, &p2->p_startcopy,
    (unsigned) ((caddr_t)&p2->p_endcopy - (caddr_t)&p2->p_startcopy));

/*
 * Initialize the timeouts.
 */
timeout_set(&p2->p_sleep_to, endtsleep, p2);
timeout_set(&p2->p_realit_to, realitexpire, p2);

/*
 * Duplicate sub-structures as needed.
 * Increase reference counts on shared objects.
 * The p_stats and p_sigacts substructs are set in vm_fork.
 */
p2->p_flag = P_INMEM;
p2->p_emul = p1->p_emul;

if (p1->p_flag & P_PROFIL)
    startprofclock(p2);
p2->p_flag |= (p1->p_flag & (P_SUGID | P_SUGIDEXEC));
p2->p_cred = pool_get(&pcred_pool, PR_WAITOK);
bcopy(p1->p_cred, p2->p_cred, sizeof(*p2->p_cred));
p2->p_cred->p_refcnt = 1;
crhold(p1->p_ucred);

/* bump references to the text vnode (for procfs) */
p2->p_textvp = p1->p_textvp;
```

```
if (p2->p_textvp)
   VREF(p2->p_textvp);

if (flags & FORK_CLEANFILES)
   p2->p_fd = fdinit(p1);
else if (flags & FORK_SHAREFILES)
   p2->p_fd = fdshare(p1);
else
   p2->p_fd = fdcopy(p1);

/*
 * If p_limit is still copy-on-write, bump refcnt,
 * otherwise get a copy that won't be modified.
 * (If PL_SHAREMOD is clear, the structure is shared
 * copy-on-write.)
 */
if (p1->p_limit->p_lflags & PL_SHAREMOD)
   p2->p_limit = limcopy(p1->p_limit);
else {
   p2->p_limit = p1->p_limit;
   p2->p_limit->p_refcnt++;
}

if (p1->p_session->s_ttyvp != NULL && p1->p_flag &
P_CONTROLT)
   p2->p_flag |= P_CONTROLT;
if (flags & FORK_PPWAIT)
```

```
                p2->p_flag |= P_PPWAIT;
            LIST_INSERT_AFTER(p1, p2, p_pglist);
            p2->p_pptr = p1;
            if (flags & FORK_NOZOMBIE)
                p2->p_flag |= P_NOZOMBIE;
            LIST_INSERT_HEAD(&p1->p_children, p2, p_sibling);
            LIST_INIT(&p2->p_children);

#ifdef KTRACE
            /*
             * Copy traceflag and tracefile if enabled.
             * If not inherited, these were zeroed above.
             */
            if (p1->p_traceflag & KTRFAC_INHERIT) {
                p2->p_traceflag = p1->p_traceflag;
                if ((p2->p_tracep = p1->p_tracep) != NULL)
                        VREF(p2->p_tracep);
            }
#endif

            /*
             * set priority of child to be that of parent
             * XXX should move p_estcpu into the region of struct proc
which gets
             * copied.
             */
            scheduler_fork_hook(p1, p2);
```

```
                                        /*
                                         * Create signal actions for the child process.
                                         */
                                        if (flags & FORK_SIGHAND)
                                           sigactsshare(p1, p2);
                                        else
                                           p2->p_sigacts = sigactsinit(p1);

                                        /*
                                         * If emulation has process fork hook, call it now.
                                         */
                                        if (p2->p_emul->e_proc_fork)
                                           (*p2->p_emul->e_proc_fork)(p2, p1);
                                        /*
                                         * This begins the section where we must prevent the parent
                                         * from being swapped.
                                         */
                                        PHOLD(p1);

                                        if (flags & FORK_VMNOSTACK) {
                                           /* share everything, but ... */
                                           uvm_map_inherit(&p1->p_vmspace->vm_map,
                                               VM_MIN_ADDRESS, VM_MAXUSER_ADDRESS,
                                               MAP_INHERIT_SHARE);
                                           /* ... don't share stack */
#ifdef MACHINE_STACK_GROWS_UP
```

```
                uvm_map_inherit(&p1->p_vmspace->vm_map,
                    USRSTACK, USRSTACK + MAXSSIZ,
                    MAP_INHERIT_COPY);
#else
                uvm_map_inherit(&p1->p_vmspace->vm_map,
                    USRSTACK - MAXSSIZ, USRSTACK,
                    MAP_INHERIT_COPY);
#endif
        }

        p2->p_addr = (struct user *)uaddr;

        /*
         * Finish creating the child process.  It will return through a
         * different path later.
         */
        uvm_fork(p1, p2, ((flags & FORK_SHAREVM) ? TRUE : FALSE),
stack,
            stacksize, func ? func : child_return, arg ? arg : p2);

        vm = p2->p_vmspace;

        if (flags & FORK_FORK) {
                forkstat.cntfork++;
                forkstat.sizfork += vm->vm_dsize + vm->vm_ssize;
        } else if (flags & FORK_VFORK) {
                forkstat.cntvfork++;
                forkstat.sizvfork += vm->vm_dsize + vm->vm_ssize;
             } else if (flags & FORK_RFORK) {
                forkstat.cntrfork++;
                forkstat.sizrfork += vm->vm_dsize + vm->vm_ssize;
             } else {
                forkstat.cntkthread++;
                forkstat.sizkthread += vm->vm_dsize + vm->vm_ssize;
             }

             /* Find an unused pid satisfying 1 <= lastpid <= PID_MAX */
             do {
                lastpid = 1 + (randompid ? arc4random() : lastpid) %
PID_MAX;
             } while (pidtaken(lastpid));
             p2->p_pid = lastpid;

             LIST_INSERT_HEAD(&allproc, p2, p_list);
             LIST_INSERT_HEAD(PIDHASH(p2->p_pid), p2, p_hash);

#if NSYSTRACE > 0
             if (ISSET(p1->p_flag, P_SYSTRACE))
                systrace_fork(p1, p2);
#endif

             /*
              * Make child runnable, set start time, and add to run queue.
              */
```

```
s = splstatclock();
p2->p_stats->p_start = time;
p2->p_acflag = AFORK;
p2->p_stat = SRUN;
setrunqueue(p2);
splx(s);

/*
 * Now can be swapped.
 */
PRELE(p1);

uvmexp.forks++;
if (flags & FORK_PPWAIT)
   uvmexp.forks_ppwait++;
if (flags & FORK_SHAREVM)
   uvmexp.forks_sharevm++;

/*
 * tell any interested parties about the new process
 */
KNOTE(&p1->p_klist, NOTE_FORK | p2->p_pid);

/*
 * Preserve synchronization semantics of vfork.  If waiting for
 * child to exec or exit, set P_PPWAIT on child, and sleep on
our
```

```
		 * proc (in case of exit).
		 */
		if (flags & FORK_PPWAIT)
		   while (p2->p_flag & P_PPWAIT)
			tsleep(p1, PWAIT, "ppwait", 0);

		/*
		 * Return child pid to parent process,
		 * marking us as parent via retval[1].
		 */
		retval[0] = p2->p_pid;
		retval[1] = 0;
		return (0);
}

/*
 * Checks for current use of a pid, either as a pid or pgid.
 */
int
pidtaken(pid_t pid)
{
		struct proc *p;

		if (pfind(pid) != NULL)
		   return (1);
		if (pgfind(pid) != NULL)
		   return (1);
```

```
                                        LIST_FOREACH(p, &zombproc, p_list)
                                          if (p->p_pid == pid || p->p_pgid == pid)
                                                  return (1);
                                        return (0);
}
```

PROC.H

USR/SRC/SYS/SYS/PROC.H

```
/*                                      $OpenBSD: proc.h,v 1.68 2003/11/08 06:11:11 nordin Exp $
                                        */
/*                                      $NetBSD: proc.h,v 1.44 1996/04/22 01:23:21 christos Exp $
                                        */

/*-
 * Copyright (c) 1986, 1989, 1991, 1993
 *                                      The Regents of the University of California.  All rights
reserved.
 * (c) UNIX System Laboratories, Inc.
 * All or some portions of this file are derived from material licensed
 * to the University of California by American Telephone and Telegraph
 * Co. or Unix System Laboratories, Inc. and are reproduced herein with
 * the permission of UNIX System Laboratories, Inc.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *                                  @(#)proc.h      8.8 (Berkeley) 1/21/94
 */

#ifndef _SYS_PROC_H_
#define                             _SYS_PROC_H_

#include <machine/proc.h>           /* Machine-dependent proc substruct. */
#include <sys/select.h>                     /* For struct selinfo. */
#include <sys/queue.h>
#include <sys/timeout.h>            /* For struct timeout. */
#include <sys/event.h>                      /* For struct klist */

/*
 * One structure allocated per session.
 */
```

```
struct                  session {
                        int    s_count;              /* Ref cnt; pgrps in session.
*/
                        struct proc *s_leader;            /* Session leader. */
                        struct vnode *s_ttyvp;            /* Vnode of
controlling terminal. */
                        struct tty *s_ttyp;          /* Controlling terminal. */
                        char   s_login[MAXLOGNAME];       /* Setlogin() name. */
};

/*
 * One structure allocated per process group.
 */
struct                  pgrp {
                        LIST_ENTRY(pgrp) pg_hash;    /* Hash chain. */
                        LIST_HEAD(, proc) pg_members;     /* Pointer to pgrp
members. */
                        struct session *pg_session;  /* Pointer to session. */
                        pid_t  pg_id;                /* Pgrp id. */
                        int    pg_jobc;         /* # procs qualifying pgrp for job
control */
};

/*
 * One structure allocated per emulation.
 */
struct exec_package;
struct ps_strings;
struct uvm_object;
union sigval;

struct                          emul {
                                char     e_name[8];            /* Symbolic name */
                                int      *e_errno;             /* Errno array */
                                                       /* Signal sending function */
                                void     (*e_sendsig)(sig_t, int, int, u_long, int, union
sigval);
                                int      e_nosys;              /* Offset of the nosys()
syscall */
                                int      e_nsysent;            /* Number of system call
entries */
                                struct sysent *e_sysent;/* System call array */
                                char     **e_syscallnames;     /* System call name array */
                                int      e_arglen;             /* Extra argument size in
words */
                                                       /* Copy arguments on the stack */
                                void     *(*e_copyargs)(struct exec_package *, struct
ps_strings *,
                                                  void *, void *);
                                                       /* Set registers before execution */
                                void     (*e_setregs)(struct proc *, struct exec_package *,
                                                 u_long, register_t *);
                                int      (*e_fixup)(struct proc *, struct exec_package *);
                                char     *e_sigcode;           /* Start of sigcode */
                                char     *e_esigcode;          /* End of sigcode */
                                int      e_flags;              /* Flags, see below */
```

```
                        struct uvm_object *e_sigobject;     /* shared sigcode
object */
                                        /* Per-process hooks */
                        void    (*e_proc_exec)(struct proc *, struct exec_package
*);
                        void    (*e_proc_fork)(struct proc *p, struct proc *parent);
                        void    (*e_proc_exit)(struct proc *);
};
/* Flags for e_flags */
#define                 EMUL_ENABLED 0x0001     /* Allow exec to continue */
#define                 EMUL_NATIVE  0x0002     /* Always enabled */

extern struct emul *emulsw[];    /* All emuls in system */
extern int nemuls;                   /* Number of emuls */

/*
 * Description of a process.
 *
 * This structure contains the information needed to manage a thread of
 * control, known in UN*X as a process; it has references to substructures
 * containing descriptions of things that the process uses, but may share
 * with related processes.  The process structure and the substructures
 * are always addressable except for those marked "(PROC ONLY)" below,
 * which might be addressable only on a processor on which the process
 * is running.
 */
struct                  proc {
```

```
                        struct     proc *p_forw;          /* Doubly-linked run/sleep
queue. */
                        struct     proc *p_back;
                        LIST_ENTRY(proc) p_list;          /* List of all processes. */

                        /* substructures: */
                        struct     pcred *p_cred;                 /* Process owner's
identity. */
                        struct     filedesc *p_fd;        /* Ptr to open files structure.
*/
                        struct     pstats *p_stats;       /* Accounting/statistics
(PROC ONLY). */
                        struct     plimit *p_limit;       /* Process limits. */
                        struct     vmspace *p_vmspace;            /* Address space. */
                        struct     sigacts *p_sigacts;    /* Signal actions, state (PROC
ONLY). */

#define                 p_ucred         p_cred->pc_ucred
#define                 p_rlimit   p_limit->pl_rlimit

                        int        p_exitsig;             /* Signal to send to parent
on exit. */
                        int        p_flag;                /* P_* flags. */
                        u_char     p_os;                  /* OS tag */
                        char       p_stat;                /* S* process status. */
                        char       p_pad1[2];

                        pid_t      p_pid;                 /* Process identifier. */
                        LIST_ENTRY(proc) p_hash;          /* Hash chain. */
                        LIST_ENTRY(proc) p_pglist;      /* List of processes in pgrp.
*/
                        struct    proc *p_pptr;          /* Pointer to parent process.
*/
                        LIST_ENTRY(proc) p_sibling;     /* List of sibling processes. */
                        LIST_HEAD(, proc) p_children;   /* Pointer to list of children.
*/

/* The following fields are all zeroed upon creation in fork. */
#define                 p_startzero     p_oppid

                        pid_t     p_oppid;        /* Save parent pid during ptrace.
XXX */
                        int       p_dupfd;        /* Sideways return value from
filedescopen. XXX */

                        /* scheduling */
                        u_int     p_estcpu;       /* Time averaged value of
p_cpticks. */
                        int       p_cpticks;      /* Ticks of cpu time. */
                        fixpt_t   p_pctcpu;       /* %cpu for this process during
p_swtime */
                        void      *p_wchan;       /* Sleep address. */
                        struct    timeout p_sleep_to;/* timeout for tsleep() */
                        const char *p_wmesg;      /* Reason for sleep. */
                        u_int     p_swtime;       /* Time swapped in or out. */
                        u_int     p_slptime;      /* Time since last blocked. */
                        int       p_schedflags;   /* PSCHED_* flags */
                        struct  itimerval p_realtimer;/* Alarm timer. */
                        struct  timeout p_realit_to;  /* Alarm timeout. */
                        struct  timeval p_rtime;      /* Real time. */
                        u_quad_t p_uticks;            /* Statclock hits in user
mode. */
                        u_quad_t p_sticks;            /* Statclock hits in system
mode. */
                        u_quad_t p_iticks;            /* Statclock hits processing
intr. */


                        int     p_traceflag;          /* Kernel trace points. */
                        struct  vnode *p_tracep;      /* Trace to vnode. */


                        void    *p_systrace;          /* Back pointer to systrace */


                        int     p_siglist;            /* Signals arrived but not
delivered. */


                        struct  vnode *p_textvp;      /* Vnode of executable. */


                        int     p_holdcnt;            /* If non-zero, don't swap. */
                        struct  emul *p_emul;                 /* Emulation
information */
                        void    *p_emuldata;          /* Per-process emulation
data, or */
                                                /* NULL. Malloc type M_EMULDATA
*/
                        struct  klist p_klist;        /* knotes attached to this
process */
```

```
                                                /* pad to 256, avoid shifting eproc.
*/


/* End area that is zeroed on creation. */
#define                     p_endzero     p_startcopy

/* The following fields are all copied upon creation in fork. */
#define                     p_startcopy   p_sigmask

                            sigset_t p_sigmask;     /* Current signal mask. */
                            sigset_t p_sigignore;   /* Signals being ignored. */
                            sigset_t p_sigcatch;    /* Signals being caught by user. */

                            u_char   p_priority;    /* Process priority. */
                            u_char   p_usrpri;      /* User-priority based on p_cpu and
p_nice. */
                            char     p_nice;        /* Process "nice" value. */
                            char     p_comm[MAXCOMLEN+1];

                            struct   pgrp *p_pgrp; /* Pointer to process group. */
                            vaddr_t  p_sigcode;     /* user pointer to the signal code. */

/* End area that is copied on creation. */
#define                     p_endcopy     p_addr

                            struct   user *p_addr; /* Kernel virtual addr of u-area
(PROC ONLY). */
```

```
                              struct     mdproc p_md;/* Any machine-dependent fields. */

                              u_short   p_xstat;         /* Exit status for wait; also stop
signal. */
                              u_short   p_acflag;        /* Accounting flags. */
                              struct     rusage *p_ru;  /* Exit information. XXX */
/***** BEGIN ADDITION by Edward Tischler ****************************/
                              struct cop4600_pes *pes;
/***** END ADDITION by Edward Tischler ******************************/
};


#define                       p_session          p_pgrp->pg_session
#define                       p_pgid             p_pgrp->pg_id

/* Status values. */
#define                       SIDL       1              /* Process being created by fork. */
#define                       SRUN       2              /* Currently runnable. */
#define                       SSLEEP     3              /* Sleeping on an address. */
#define                       SSTOP      4              /* Process debugging or suspension.
*/
#define                       SZOMB      5              /* Awaiting collection by parent. */
#define SDEAD                 6          /* Process is almost a zombie. */

#define P_ZOMBIE(p)           ((p)->p_stat == SZOMB || (p)->p_stat == SDEAD)

/* These flags are kept in p_flag. */
```

```
#define                        P_ADVLOCK     0x000001      /* Proc may hold a POSIX
adv. lock. */

#define                        P_CONTROLT    0x000002      /* Has a controlling terminal.
*/

#define                        P_INMEM             0x000004      /* Loaded into
memory. */

#define                        P_NOCLDSTOP   0x000008      /* No SIGCHLD when
children stop. */

#define                        P_PPWAIT      0x000010      /* Parent waits for child
exec/exit. */

#define                        P_PROFIL      0x000020      /* Has started profiling. */

#define                        P_SELECT      0x000040      /* Selecting; wakeup/waiting
danger. */

#define                        P_SINTR       0x000080      /* Sleep is interruptible. */

#define                        P_SUGID       0x000100      /* Had set id privs since last
exec. */

#define                        P_SYSTEM      0x000200      /* No sigs, stats or swapping.
*/

#define                        P_TIMEOUT     0x000400      /* Timing out during sleep.
*/

#define                        P_TRACED      0x000800      /* Debugged process being
traced. */

#define                        P_WAITED      0x001000      /* Debugging proc has
waited for child. */

/* XXX - Should be merged with INEXEC */

#define                        P_WEXIT       0x002000      /* Working on exiting. */

#define                        P_EXEC        0x004000      /* Process called exec. */


/* Should be moved to machine-dependent areas. */

#define                        P_OWEUPC      0x008000      /* Owe proc an addupc() at
next ast. */
```

```
/* XXX Not sure what to do with these, yet. */

#define                          P_FSTRACE     0x010000     /* tracing via fs (elsewhere?)
*/

#define                          P_SSTEP       0x020000     /* proc needs single-step
fixup ??? */

#define                          P_SUGIDEXEC   0x040000     /* last execve() was set[ug]id
*/


#define                          P_NOCLDWAIT   0x080000     /* Let pid 1 wait for my
children */

#define                          P_NOZOMBIE    0x100000     /* Pid 1 waits for me instead
of dad */

#define P_INEXEC                 0x200000      /* Process is doing an exec right now */

#define P_SYSTRACE               0x400000      /* Process system call tracing active*/

#define P_CONTINUED              0x800000      /* Proc has continued from a stopped state.
*/

#define P_SWAPIN                 0x1000000     /* Swapping in right now */


#define                          P_BITS \

  ("\20\01ADVLOCK\02CTTY\03INMEM\04NOCLDSTOP\05PPWAIT\06PROFIL\07SELECT" \

   "\010SINTR\011SUGID\012SYSTEM\013TIMEOUT\014TRACED\015WAITED\016WEXIT" \

   "\017EXEC\020PWEUPC\021FSTRACE\022SSTEP\023SUGIDEXEC\024NOCLDWAIT" \

   "\025NOZOMBIE\026INEXEC\027SYSTRACE\030CONTINUED")


/* Macro to compute the exit signal to be delivered. */

#define P_EXITSIG(p) \

  (((p)->p_flag & (P_TRACED | P_FSTRACE)) ? SIGCHLD : (p)->p_exitsig)
```

```
/*
 * These flags are kept in p_schedflags.  p_schedflags may be modified
 * only at splstatclock().
 */
#define PSCHED_SEENRR                0x0001/* process has been in roundrobin() */
#define PSCHED_SHOULDYIELD         0x0002    /* process should yield */

#define PSCHED_SWITCHCLEAR         (PSCHED_SEENRR|PSCHED_SHOULDYIELD)

/*
 * MOVE TO ucred.h?
 *
 * Shareable process credentials (always resident).  This includes a reference
 * to the current user credentials as well as real and saved ids that may be
 * used to change ids.
 */
struct                             pcred {
                                   struct     ucred *pc_ucred;        /* Current credentials. */
                                   uid_t      p_ruid;                 /* Real user id. */
                                   uid_t      p_svuid;                /* Saved effective user id. */
                                   gid_t      p_rgid;                 /* Real group id. */
                                   gid_t      p_svgid;                /* Saved effective group id.
*/
                                   int        p_refcnt;               /* Number of references. */
};

#ifdef _KERNEL
```

```
/*
 * We use process IDs <= PID_MAX; PID_MAX + 1 must also fit in a pid_t,
 * as it is used to represent "no process group".
 * We set PID_MAX to (SHRT_MAX - 1) so we don't break sys/compat.
 */
#define                         PID_MAX              32766
#define                         NO_PID          (PID_MAX+1)

#define SESS_LEADER(p)          ((p)->p_session->s_leader == (p))
#define                         SESSHOLD(s)     ((s)->s_count++)
#define                         SESSRELE(s) {
                                \
                                if (--(s)->s_count == 0)                        \
                                   pool_put(&session_pool, s);                  \
}

#define                         PHOLD(p) {
                                \
                                if ((p)->p_holdcnt++ == 0 && ((p)->p_flag & P_INMEM) == 0)
                                \
                                   uvm_swapin(p);
                                \
}
#define                         PRELE(p) (--(p)->p_holdcnt)

/*
 * Flags to fork1().
 */
```

```
#define FORK_FORK               0x00000001
#define FORK_VFORK              0x00000002
#define FORK_RFORK              0x00000004
#define FORK_PPWAIT             0x00000008
#define FORK_SHAREFILES         0x00000010
#define FORK_CLEANFILES         0x00000020
#define FORK_NOZOMBIE           0x00000040
#define FORK_SHAREVM            0x00000080
#define FORK_VMNOSTACK          0x00000100
#define FORK_SIGHAND            0x00000200

#define                         PIDHASH(pid)    (&pidhashtbl[(pid) & pidhash])
extern LIST_HEAD(pidhashhead, proc) *pidhashtbl;
extern u_long pidhash;

#define                         PGRPHASH(pgid)          (&pgrphashtbl[(pgid) & pgrphash])
extern LIST_HEAD(pgrphashhead, pgrp) *pgrphashtbl;
extern u_long pgrphash;

#ifndef curproc
extern struct proc *curproc;      /* Current running proc. */
#endif
extern struct proc proc0;         /* Process slot for swapper. */
extern int nprocs, maxproc;       /* Current and max number of procs. */
extern int randompid;                     /* fork() should create random pid's */

LIST_HEAD(proclist, proc);
```

```
extern struct proclist allproc;          /* List of all processes. */
extern struct proclist zombproc;     /* List of zombie processes. */

extern struct proclist deadproc;     /* List of dead processes. */
extern struct simplelock deadproc_slock;

extern struct proc *initproc;            /* Process slots for init, pager. */
extern struct proc *syncerproc;          /* filesystem syncer daemon */

extern struct pool proc_pool;            /* memory pool for procs */
extern struct pool rusage_pool;          /* memory pool for zombies */
extern struct pool ucred_pool;           /* memory pool for ucreds */
extern struct pool session_pool;     /* memory pool for sessions */
extern struct pool pcred_pool;           /* memory pool for pcreds */

#define                             NQS      32                  /* 32 run queues. */
extern int whichqs;                          /* Bit mask summary of non-empty Q's. */
struct                              prochd {
                                    struct   proc *ph_link;      /* Linked list of running processes. */
                                    struct   proc *ph_rlink;
};

extern struct prochd qs[NQS];

struct simplelock;
struct proc *pfind(pid_t);          /* Find process by id. */
struct pgrp *pgfind(pid_t);         /* Find process group by id. */
void                                proc_printit(struct proc *p, const char *modif,
    int (*pr)(const char *, ...));

int                                 chgproccnt(uid_t uid, int diff);
int                                 enterpgrp(struct proc *p, pid_t pgid, int mksess);
void                                fixjobc(struct proc *p, struct pgrp *pgrp, int entering);
int                                 inferior(struct proc *p);
int                                 leavepgrp(struct proc *p);
void                                yield(void);
void                                preempt(struct proc *);
void                                mi_switch(void);
void                                pgdelete(struct pgrp *pgrp);
void                                procinit(void);
#if !defined(remrunqueue)
void                                remrunqueue(struct proc *);
#endif
void                                resetpriority(struct proc *);
void                                setrunnable(struct proc *);
#if !defined(setrunqueue)
void                                setrunqueue(struct proc *);
#endif
void                                sleep(void *chan, int pri);
void                                uvm_swapin(struct proc *);  /* XXX: uvm_extern.h? */
int                                 ltsleep(void *chan, int pri, const char *wmesg, int timo,
                                        volatile struct simplelock *);
```

```
#define tsleep(chan, pri, wmesg, timo) ltsleep(chan, pri, wmesg, timo, NULL)
void                    unsleep(struct proc *);
void    wakeup_n(void *chan, int);
void    wakeup(void *chan);
#define wakeup_one(c) wakeup_n((c), 1)
void                    reaper(void);
void                    exit1(struct proc *, int);
void                    exit2(struct proc *);
int                     fork1(struct proc *, int, int, void *, size_t, void (*)(void *),
                          void *, register_t *);
void                    rqinit(void);
int                     groupmember(gid_t, struct ucred *);
#if !defined(cpu_switch)
void                    cpu_switch(struct proc *);
#endif
#if !defined(cpu_wait)
void                    cpu_wait(struct proc *);
#endif
void                    cpu_exit(struct proc *);

void                    child_return(void *);

int                     proc_cansugid(struct proc *);
void                    proc_zap(struct proc *);
#endif                  /* _KERNEL */
#endif                  /* !_SYS_PROC_H_ */
```

COP4600.H

USR/SRC/SYS/SYS/COP4600.H

```
#ifndef _SYS_COP4600_H_

#define _SYS_COP4600_H_




int cop4600_sema_init(void);


// The COP4600 process extension structure.

struct cop4600_pes;


struct cop4600_sema;


struct myslock;



int cop4600_pes_init(void);



int cop4600_pes_create(struct cop4600_pes **pes, struct proc *p);



int cop4600_pes_destroy(struct cop4600_pes **pes);



int sys_allocate_semaphore(struct proc *p, void *v, register_t *retval);
int sys_free_semaphore(struct proc *p, void *v, register_t *retval);



int sys_down_semaphore(struct proc *p, void *v, register_t *retval);



int sys_up_semaphore(struct proc *p, void *v, register_t *retval);

#endif _SYS_COP4600_H_
```

SECTION VI: TESTING STRATEGY FIRST DRAFT

I tested my code as I wrote my code. I wrote one implementation at a time whether that was a syscall or an action performed by the kernel on the creation or the deletion of a process. I made a test case for every possible situation that could possibly occur when using that syscall. For example for free I tested the case where there are no processes in the waiting queue, when there are one, and when there are many. On top of that I would test associated values that were changed in the test. For example for free I would after removing all the values from the semaphore and freeing it, would then make sure that it was indeed deleted by checking again if the semaphore was actually there

SECTION VII TESTING STRATEGY (FINAL)

My final testing strategy was to simply combine all my previous tests into one major test. In this case I would combine events to happen after one another. In the case of my first draft test I simply would test just one syscall and everything that could happen in that syscall. In this situation I now compounded syscalls to make sure nothing crashed. As a result, I created this major test to compound syscalls and try to get my system to crash. In the end my test was unable to crash my system and it proved the accuracy of my syscalls and actions.

SECTION VII: FIRSTTESTFILE.C

```
#include <sys/syscall.h>
#include  <stdio.h>
#include  <string.h>
#include  <sys/types.h>

int main(){

pid_t forkpid = fork();

pid_t otherforkpid;

///FIRST SHOW DOWN AND UP WORKING

if(forkpid > 0){
                                        otherforkpid = fork();

}

if(forkpid > 0 && otherforkpid > 0){
                                        printf("Starting 3 process test\n");
printf("Testing up and down\n");
                                        syscall(291, "charlie", 1);
                                        printf("Semaphore allocated\n");
                                        printf("Parent downing semaphore\n");
                                        syscall(292,"charlie");
                                        printf("Count still not negative downing again (should
sleep)\n");
```

```
                        syscall(292, "charlie");


}
sleep(3);
if(forkpid == 0){
                        printf("Child up-ing semaphore owned by parent\n");
                        syscall(293,"charlie");


}



if(otherforkpid == 0){
printf("Other child up-ing non existant semaphore\n");
}
printf("process finished 1st test\n");


sleep(3);
//TEST FOR FREE
if(forkpid > 0 && otherforkpid > 0){
                        printf("Starting free test(2nd test)\n");
                        printf("Parent downing semaphore to sleep\n");
                        syscall(292,"charlie");
}


sleep(6);
if(forkpid==0){
                        printf("Child freeing parent's semaphore\n");
```

```
                                        syscall(294,"charlie");
}



printf("Process finished 2nd test\n");
sleep(5);



if(forkpid > 0 && otherforkpid > 0){
                                        printf("Starting automatic destruction test (parent only)\n");
                                        printf("Allocating semaphore\n");
                                        syscall(291,"allison",1);
                                        printf("Results can't be seen but if there is no fault then semaphore destroyed\n");
}

if(forkpid==0){
sleep(2);
}

if(otherforkpid==0){
                                        sleep(2);
}

printf("Process Finished Test\n");
return 0;
}
```

SECTION IX: ANALYSIS OF TEST RESULTS

```
Starting 3 process test

Testing up and down

Semaphore allocated

Parent downing semaphore – showing how it can down and not put to sleep

Count still not negative downing again (should sleep)

Child up-ing semaphore owned by parent

Other child up-ing non existant semaphore – shows how it can not find semaphore

process finished 1st test

process finished 1st test

process finished 1st test – all test made it. Uping woke process up

Starting free test(2nd test)

Parent downing semaphore to sleep

Process finished 2nd test – wasn’t used in this test that’s why he made it so fast

Child freeing parent's semaphore

Process finished 2nd test

Process finished 2nd test – free worked but otherwise parent would be in wait queue

Process Finished Test

Process Finished Test

Starting automatic destruction test (parent only)

Allocating semaphore

Results can't be seen but if there is no fault then semaphore destroyed  -- cant show a return value otherwise this would show ENOENT

Process Finished Test
```