## Steps (by file) for kernel Threads project: clone() and join()

**In user.h:**

**A computer screen with green and white text

AI-generated content may be incorrect.**

**In proc.h:**

int clone(void (\*fcn)(void\*, void\*), void \*arg1, void \*arg2, void \*stack);

A screenshot of a computer

AI-generated content may be incorrect.

inside of struct proc in proc.h, add:

void \*stack;

A screenshot of a computer program

AI-generated content may be incorrect.

**In usys.S:**

add system calls SYSCALL(join) and SYSCALL(clone)

A computer screen shot of text

AI-generated content may be incorrect.

**In syscall.c:**

add extern int sys\_clone(void); and extern int sys\_join(void); near the top

**A screenshot of a computer

AI-generated content may be incorrect.**

then add the system calls to the syscalls array

**A black background with white text

AI-generated content may be incorrect.**

**In sysproc.c**

add the following to the top of the file, after the imports

extern int join(void \*\*);

int

sys\_join(void)

{

void \*stackp;

if (argptr(0, (char\*\*)&stackp, sizeof(void \*)) < 0)

return -1;

return join((void \*\*)stackp);

}

int

sys\_clone(void)

{

uint fcn, arg1, arg2, stack;

if(argint(0, (int\*)&fcn) < 0 || argint(1, (int\*)&arg1) < 0 || argint(2, (int\*)&arg2) < 0 || argint(3, (int\*)&stack) < 0)

return -1;

return clone((void (\*)(void\*, void\*))fcn, (void\*)arg1, (void\*)arg2, (void\*)stack);

}

A computer screen shot of a black screen

AI-generated content may be incorrect.

**In syscall.h:**

define SYS\_join and SYS\_clone with unique numbers

A screenshot of a computer

AI-generated content may be incorrect.

**In proc.c:**

put clone() code into the bottom of proc.c:

int clone(void (\*fcn)(void\*, void\*), void \*arg1, void \*arg2, void \*stack) {

int i;

struct proc \*np;

struct proc \*curproc = myproc();

// Stack must be page-aligned and non-null

if ((uint)stack % PGSIZE != 0 || stack == 0)

return -1;

// Allocate process (thread)

if ((np = allocproc()) == 0)

return -1;

// Share address space

np->pgdir = curproc->pgdir;

np->sz = curproc->sz;

np->parent = curproc;

// Copy trapframe from parent

\*np->tf = \*curproc->tf;

//So child sees 0 return from syscall

np->tf->eax = 0;

// Set up new user stack

uint sp = (uint)stack;

// Push arg2

sp -= 4;

\*(uint\*)sp = (uint)arg2;

// Push arg1

sp -= 4;

\*(uint\*)sp = (uint)arg1;

// Push return address

sp -= 4;

\*(uint\*)sp = (uint)exit;

// Set the stack pointer for the child process to the top of the stack (where args are located)

np->tf->esp = sp;

// Set instruction pointer to thread function, which is where the child will start executing

np->tf->eip = (uint)fcn;

// Save the user stack for the join function

np->stack = stack;

// Share open files with the child

for(i = 0; i < NOFILE; i++) {

if(curproc->ofile[i])

np->ofile[i] = filedup(curproc->ofile[i]);

}

np->cwd = idup(curproc->cwd);

safestrcpy(np->name, curproc->name, sizeof(curproc->name));

// Mark the new process as runnable

np->state = RUNNABLE;

return np->pid;

}

A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a black screen

AI-generated content may be incorrect.

at the top of proc.c, initialize join() with:

int join(void \*\*);

In proc.c, inside of the allocproc() function, add this right under the //Allocate Kernel Stack loop:

p->stack = 0;

A computer screen with white text

AI-generated content may be incorrect.

At the bottom of proc.c, under clone(), add join():

int

join(void \*\*stackp)

{

struct proc \*p;

int havekids;

struct proc \*curproc = myproc();

acquire(&ptable.lock);

for(;;) {

havekids = 0;

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if(p->parent != curproc || p->pgdir != curproc->pgdir)

continue;

havekids = 1;

if(p->state == ZOMBIE){

int pid = p->pid;

if (stackp)

\*stackp = p->stack; // Retrieve the saved user stack pointer

// Free resources (like wait())

kfree(p->kstack);

p->kstack = 0;

p->state = UNUSED;

p->pid = 0;

p->parent = 0;

p->name[0] = 0;

p->killed = 0;

p->stack = 0; // Clear saved stack pointer

release(&ptable.lock);

return pid;

}

}

if(!havekids || curproc->killed){

release(&ptable.lock);

return -1;

}

sleep(curproc, &ptable.lock);

}

}

A screenshot of a computer program

AI-generated content may be incorrect.

A black screen with a black background

AI-generated content may be incorrect.

**clone\_test.c**

#include "types.h"

#include "stat.h"

#include "user.h"

#define STACK\_SIZE 4096

void thread\_func(void \*arg1, void \*arg2) {

int a = \*(int\*)arg1;

int b = \*(int\*)arg2;

printf(1, "Thread running: arg1 = %d, arg2 = %d\n", a, b); // Changed printf to use file descriptor

exit(); // No arguments in x86 xv6

}

int

main(int argc, char \*argv[])

{

void \*stack;

int arg1 = 42;

int arg2 = 24;

// Allocate a page for the user stack

stack = malloc(STACK\_SIZE);

if (stack == 0) {

printf(1, "Failed to allocate stack\n"); // Changed printf to use file descriptor

exit();

}

// Ensure page-aligned stack

if ((uint)stack % STACK\_SIZE != 0) {

stack = (void \*)((uint)stack + (STACK\_SIZE - ((uint)stack % STACK\_SIZE)));

}

// Call clone with correct arguments

int pid = clone(thread\_func, &arg1, &arg2, (void\*)((char\*)stack + STACK\_SIZE));

if (pid < 0) {

printf(1, "clone failed\n"); // Changed printf to use file descriptor

free(stack); // cleanup if clone fails

exit();

}

printf(1, "Parent: created thread with pid %d\n", pid); // Changed printf to use file descriptor

wait();

free(stack);

printf(1, "Parent: thread finished\n"); // Changed printf to use file descriptor

exit();

}

A screenshot of a computer screen

AI-generated content may be incorrect.