SMUNCH PROJECT

OPERATING SYSTEMS
CPSC822

Team:
Chinmay Joshi

Shreya Deodhar

Chirantan Sharma

Prantit Lokre

Under The Guidance of
Dr. Robert Geist

CLEMSON UNIVERSITY

System Call – SMUNCH.C

```
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/pid.h>
#include <linux/types.h>
#include <linux/syscalls.h>
#include <linux/linkage.h>
SYSCALL_DEFINE2(smunch,int,pid,unsigned long,bit_pattern)
{
        unsigned long flags;
        struct task_struct *task;
        int ret;
        rcu_read_lock();
                task = pid_task(find_vpid(pid),PIDTYPE_PID);
        rcu_read_unlock();
        if(!task) return -1;        // Process not present
        if(!lock_task_sighand(task,&flags))
        {
                unlock_task_sighand(task,&flags); //Process failes to give the lock. Either dead/dying
                return -1;
        }
        if(!thread_group_empty(task))
        {
                printk(KERN_ALERT "\nMULTI-Threaded Process, Exiting without processing");
                ret=-1; goto return_path;
        }
        printk(KERN_ALERT "\nExit State:%XH,State=%XH\n",task->exit_state,task->state);
        if(bit_pattern & (1UL<<(SIGKILL-1)) && (task->exit_state & EXIT_ZOMBIE))
        {
                printk(KERN_ALERT "\nSIGKILL present while Process is Zombie, releasing task!!");
                unlock_task_sighand(task,&flags);
                release_task(task);  // detach_pid is called from release_task()
                return 0;
        }
        printk(KERN_ALERT "!SIGKILL || (ordinary process) || DeepSleep, sending all signals!");
        /* It is Users responsility to note that signals will get handled in 1-64 order*/
        task->signal->shared_pending.signal.sig[0] = bit_pattern;
        set_tsk_thread_flag(task,TIF_SIGPENDING); // Set SIGPENDING flag
        if(task->state & TASK_UNINTERRUPTIBLE)
        {
        printk(KERN_ALERT "\nProcess is in Uniterruptible Wait-DeepSleep!!");
        wake_up_process(task); //wake_up_state(task,TASK_INTERRUPTIBLE); does not work
        ret=0; goto return_path;
        }
//      wake_up_process(task);//Ordinary Process-signal_wake_up may also work
        signal_wake_up(task,1);
        ret=0;
        return_path:
        unlock_task_sighand(task,&flags);
        return ret;
}
```

User codes for testing

1. Superkill : Sends Multiple signals at once

Usage: Superkill PID [signal_numbers…]

```
#include <stdio.h>
#include <errno.h>
#include <sys/syscall.h>
#include <stdlib.h>
#define smunch(a,b) syscall(326,a,b)  // ...or whatever number is next
void msg(void)
{
        printf("Usage: superkill pid signals_tosend");

}
int main(int argc,char *argv[])
{
        unsigned long bit_pattern=0;
        int pid,sig,i;
        if(argc<3)
        {
        msg(); return -1;
        }
        pid= atoi(argv[1]);
        for(i=2;i<argc;i++)
        {

        sig = atoi(argv[i]);
        bit_pattern=bit_pattern | 1UL << (sig-1);
        }
        printf("Killing PID=%d with BitPattern %XH\n",pid,bit_pattern);
        sig= smunch(pid,bit_pattern);
        printf("Smunch Returns %d",sig);
}
```

User Code : Creates DeepSleep Process waiting in uninterruptible state, uses earlier system call deepsleep

```
#include <stdio.h>
#include <errno.h>
#include <sys/syscall.h>
#include <sys/signal.h>
#include <unistd.h>

#define deepsleep() syscall(325)  // ...or whatever number is next
void custom1()
{
printf("Deepsleep USR1");
return;
}
void custom2()
{
```

```
printf("Deepsleep USR2");
return;
}
int main()
{
        signal(SIGUSR1,custom1);
signal(SIGUSR2,custom2);
        printf("goodnight, Irene\n");
        deepsleep();
        printf("oops ... woke up!\n");
        sleep(1000);
}
```

User Code : Creates Processes with custom handlers for SIGUSR1 , SIGUSR2

```
#include <stdio.h>
#include <errno.h>
#include <sys/syscall.h>
#include <sys/signal.h>
#include <unistd.h>
void Custom()
{
  printf("\nSIGUSR1 Custom handler\n");
        return;
}
void custom2()
{
printf("\nSIGUSR2 Custom Handler\n");
return;
}
int main()
{
  int pid, ret;
  switch(pid = fork())
  {
    case 0:
      signal(SIGUSR1, Custom);
           sleep(100);
      printf("Child process %d\n",getpid());
      break;
    default:
      printf("Parent process %d \n",getpid());
           signal(SIGUSR1,Custom);
           signal(SIGUSR2,custom2);
      //ret = kill(pid, SIGUSR1);
      sleep(1000);
           printf("kill returned %d\n", ret);
      sleep(50);
      break;
  }
  return 0;
}
```