سید نامی مدرسی
محمدرضا شهرستانی
تیر ۱۴۰۰

گزارش کار پروژه درس سیستم عامل

پیاده سازی کرنل ترد در xv6

مراحل انجام شده :

در این پروژه باید ترد ها را به کمک پردازه ها پیاده می کردیم . برای این کار نیاز است که یک پردازه را ساخته و مقادیر متغیر های کنترلی آن پردازه را جوری تنظیم کنیم که مانند یک ترد عمل کند . برای این کار باید کاری کنیم که پردازه جدید در همان فضای ادرس پردازه پدر کار کند ولی خود نیاز به یک استک جدید دارد .

پس در سیستم کال clone مشابه تابع fork عمل می کنیم ولی مقدار pgdir را برای پردازه جدید برابر مقدار پردازه پدر قرار می دهیم .

سپس استک جدید ساخته شده را هم به این تابع داده تا ترد به این استک اشاره کند تا بتواند از آن استفاده کند .

در سیستم کال join هم مشابه تابع wait عمل می کنیم منتها یک تفاوت اساسی وجود دارد اینکه وقتی یک ترد کارش تمام شد نباید تمام فضای ادرس را پاک کند چرا که در این صورت تمام ترد های دیگر و حتی پردازه اصلی تمام می شوند ! پس نیاز است که چک کنیم ایا ترد دیگری دارد از حافظه استفاده می کند یا نه که اگر استفاده می کرد صرفا موارد مربوط به همان ترد یعنی استک خودش را پاک کنیم .

در تابع thread_create از تابع کلون استفاده می کنیم یعنی یک استک ساخته و سپس سیستم کال کلون را صدا می زنیم و استک ساخته شده را به آن می دهیم و به این شکل می توانیم استفاده از سیستم کال کلون را ساده تر کنیم .

در ادامه خروجی برای تست های مختلف را مشاهده می کنیم .

به طور دقیق تر در سیستم کال clone یک پردازه جدید درست می کنیم و فیلد های کنترلی آن را جوری مقدار دهی می کنیم که مانند ترد عمل کند مثلا اینکه این پردازه باید به همان فضای ادرسی اشاره کند که پردازه پدر اشاره می کند .

منظور از فیلد های کنترلی موارد موجود در ساختار پردازه ها در این سیستم عامل است :

```
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  uint thread_stack_address;   // address of thread stack
  uint threads;                // thread number
};
```

سپس در سیستم کال join مانند wait عمل کرده اما با یک سری شرط های بیشتر مثلا اینکه ما باید ترد ها را شناسایی کنیم پس به کمک دو حلقه تو در تو می توانیم ترد های هر پردازه را پیدا کنیم (آن هایی که فضای ادرس یکسان دارند) و اگر آن ترد زامبی شده باشد آن را تمام می کنیم در عین حال که پردازه پدر منتظر می ماند تا ترد ها اجرا بشوند .

در تابع thread_create نیز موارد لازم برای clone را می سازیم و آن را صدا می زنیم یعنی استک لازم برای آن و به این شکل کار را برای استفاده از clone برای کاربر ساده می کنیم .

نتایج :

تست زیر یک تست بسیار ساده است و صرفا قرار است یک ترد ساخته بشود و یک تابع که فقط قرار است کاراکتر # را چاپ کند اجرا کند . در تصویر زیر مشاهده می کنیم که این ترد این تابع را اجرا کرده است و pid مربوط به این ترد هم 8 است هم در clone و هم در join پس یعنی برنامه به درستی ترد تمام شده را حذف می کند.

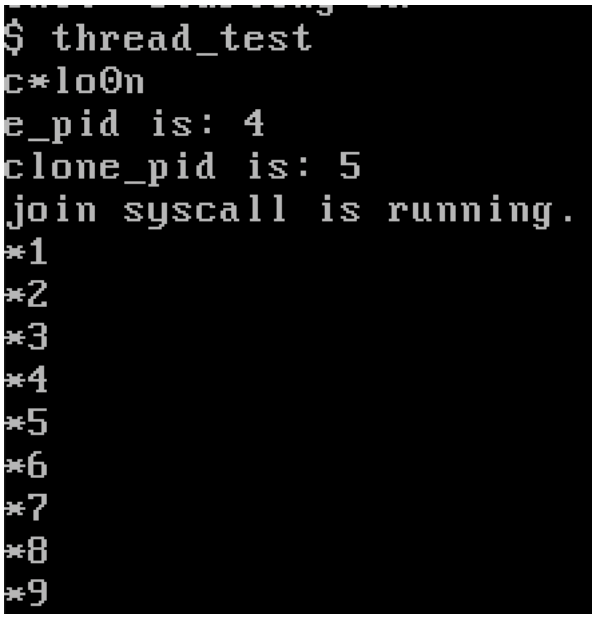
```
$ thread_test
#clone_pid is: 8
join syscall is running.
join_pid is: 8
```

تست زیر یک تست دیگر است که در این تست یک متغیر x را به یک تابع ورودی می دهیم و در تابع جمع اعداد 1 تا x را محاسبه می کنیم و در نهایت در ترد main آن را چاپ می کنیم .

مشاهده می کنیم که تابع برای مقدار x = 5 به درستی خروجی 15 را داده است .

```
$ thread_test
clone_pid is: 4
join syscall is running.
sum is :15
```

تست زیر یک تست دیگر است که در این تست دو ترد ایجاد می کنیم . یک متغیر global داریم . یک یک ترد تابعی را اجرا می کند که این متغیر را یک واحد زیاد کرده و یک ترد تابعی را اجرا می کند که این متغیر را چاپ کرده و مشاهده می کنیم که هر دو ترد ساخته شده و با هم در حال کار هستند.



تست زیر یک تست دیگر است که در این تست یک ارایه داریم که شامل اعداد ۱ تا ۶ است .

۶ ترد ساخته که هر کدام یک خانه از این ارایه را یک متغیر global جمع می کند و در نهایت مقدار این متغیر را در ترد main چاپ می کنیم .

مشاهده می کنیم که خروجی 21 که درست است چاپ می شود .

```
$ thread_test
pid 4 thread_test: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
clone_pid is: 4
join syscall is running.
join_pid is: 4
pid 5 thread_test: trap 14 err 5 on cpu 0 eip 0xffffffff addr 0xffffffff--kill proc
clone_pid is: 5
join syscall is running.
join_pid is: 5
pid 6 thread_test: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
clone_pid is: 6
join syscall is running.
join_pid is: 6
pid 7 thread_test: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
clone_pid is: 7
join syscall is running.
join_pid is: 7
pid 8 thread_test: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
clone_pid is: 8
join syscall is running.
join_pid is: 8
pid 9 thread_test: trap 14 err 5 on cpu 0 eip 0xffffffff addr 0xffffffff--kill proc
clone_pid is: 9
join syscall is running.
join_pid is: 9
*21
```