GEBZE TECHNICAL UNIVERSITY
SPRING 2014-2015
CSE 312 – OPERATING SYSTEMS
PINTOS:PROJECT2
REPORT

Alican ÖZER
Selim AKSOY

---- PRELIMINARIES ----

>> If you have any preliminary comments on your submission, notes for the
>> TAs, or extra credit, please give them here.

>> Please cite any offline or online sources you consulted while
>> preparing your submission, other than the Pintos documentation, course
>> text, lecture notes, and course staff.

ARGUMENT PASSING
==========

---- DATA STRUCTURES ----

>> A1: Copy here the declaration of each new or changed `struct' or
>> `struct' member, global or static variable,functions , `typedef', or
>> enumeration. Identify the purpose of each in 25 words or less.

thread.h

```
int fd;                  // file description
tid_t parent;            // thread parent
struct list child_list; // child process list
struct list file_list;    // thread file list
struct process* cp;   // thread child process indicator
bool is_alive (int pid);  // verilen pid ile all list içerisinde arama yapar
```

thread.c

- *Verilen pid ye baglı olarak sistem uzerinde all list icerisindeki*
- *esit olan thread var ise true yoksa falsereturn eder.*
- *Bu sayede verilen pid threadin yasayıp yasamadıgı hakkında bilgi ediniriz.*
- *Bu sekılde child process işlem yaparken o an threadin ölme durumlarını kontrol etmemize yardımcı olur*

```
bool is_alive (int pid){
 struct list_elem *e;

 for (e = list_begin (&all_list); e != list_end (&all_list);
     e = list_next (e)){
    struct thread *t = list_entry (e, struct thread, allelem);
    if (t->tid == pid)
            return true;
  }
 return false;
}
```

process.c

- *load() ile verilen filename parsing ile istenilen argumanların stack uzerine*
- *eklenmesi ve diger islemler bu kısımda yapılır.*
- *token_ptr verilen file name 'i argumanlar ile birlikte point eder.*

```
static bool load (const char *cmdline, void (**eip) (void), void **esp,
                  char** token_ptr);
```

process_execute

- *Verilen string icerisinden file name 'i alırız.*
- *Bu sekılde bır thread olustururulur*

```
char *token_ptr;
file_name = strtok_r((char *) file_name, " ", &token_ptr);
```

- *thread create olurken ayrıca bu arada process olustururuz ve threadin child listesine eklerim*
- *tid = thread_create (file_name, PRI_DEFAULT, start_process, fn_copy);*

start_process

```
char *token_ptr;
file_name = strtok_r(file_name, " ", &token_ptr);
success = load (file_name, &if_.eip, &if_.esp, &token_ptr);
```

- Resign ettigim load fonksiyonu zaten sistemde cagırılıyordu.
- Sadece token_ptr parametre olarak verdim.
- load icerisinde cagırılan setup_stack uzerinde islem yapmamı saglamaktadır.

```
if (success){
        thread_current()->cp->load = SUCCESSFULLY_LOAD;
```

- *load islemi success o zaman o anki calısan threadin*
- *gosterdigi child process indicator'u de load true olarak ekledim.*

```
        }else{
```

- *bir hata olustuysa stack uzerinde o zaman load fail oldu bu sayede threadin islem yapıcak olan*
- *process olusturma durumunun neden nasıl oldugu konusunda bılgımız olucaktır.*

```
        thread_current()->cp->load = LOAD_FAIL;}
```

process_wait

- *Verilen parametre id si ile current_thread uzerindeki child listesinde*
- *bulunan child process compare edilir.*
- *wait statusune bakarak child processin*
- *durumundan dolayı exit status'une de bakmamız gerekir.*

İnt process_wait (tid_t child_tid UNUSED){
struct process* child = child_process_check(child_tid);

- *verilen tid ile o anki calısan threadte bulunan child listesinde (process listesinde)*
- *bu tid ile bulunuyormu bulunmuyorsa NULL return ettirdim bu sayede ERROR mesajı dondurebılırım.*

if (child == NULL){
return ERROR; }

- *child process in wait yapısı true ise zaten bekleme vaziyetinde yine bekleme durumuna getirilmesi yanlıs olucaktır.busy_wait yapmasın*

if (child->wait == true){
return ERROR; }

- *process herhangi sıkıntı iceren durumda degıl o zaman wait = true oldu.*

child->wait = true;

- *en sonunda bu durumlar icermiyorsa o an child process wait statusu true ettik ve artık wait statusundedir. ayrıca herhangi bir sekilde interrupt gelmedigi uzere while dongusunde bekletilir bu sayede waiting yapılmıs olucaktır.*

while (!child->exit){

- *anki threadin exit olması durumundan dolayı child process indicator'un exit statusu true olma durumunu kontrol ettim.*

barrier();}

- *process_wait islemi direk normal de cıkıyor wait olmuyor fakat exit status false oldugu surece beklenmesi gerekiyor cunkı exit degil. (İnternet uzerinde bir sitede barrier yapısı kullanarak waiting yapılır yazıyordu.Arastırma sonucunda buldum.*

int temp_status = child->status; *// o anki child processin statusu return edilecek.*
remove_process_by(child); *// child listesinden silinir.*
return temp_status;
}

process_exit

- *Process ölürken açık dosyalarını kapatır.*

close_file_by_fd(CLOSE_ALL);

- *process exit olurken tum liste de bulunan child process ler silinir.*

remove_child_processes_all();
if (is_alive(cur->parent)){

- *child process indicator 'umuz exit status 'u true yapılır.*
- *Bu sayede child processin exit statusunden exit oldugunu anlayacagız.*

cur->cp->exit = true; }

- *setup stack argumanların stack icerisine eklenmesini saglamaktadır.*
- *token_ptr argumanların da oldugu yapıyı gosterir.*
- *bu sayede stack uzerine argumanları ekleme sansımız oldu.*

static bool setup_stack (void **esp, const char* file_name, char token_ptr);**

stack

if (!setup_stack (esp, file_name, token_ptr))
goto done;

- *token_ptr icerisindeki argumanların stacke atıldı*
- *malloc ile arguman sayısı kadar yer aldım*

```
int i, argc = 0, argv_size = ARGV_COUNTER;
char *token;
char **argv = malloc(ARGV_COUNTER *sizeof(char *));

for (token = (char *) file_name; token != NULL;token = strtok_r (NULL, " ", token_ptr))
```

- *statement icerisinde stack uzerine ekleme yaptım.*
- *her ekleme isleminden sonra argc bir artırırız.*

>> A2: Briefly describe how you implemented argument parsing. How do
>> you arrange for the elements of argv[] to be in the right order?
>> How do you avoid overflowing the stack page?

- *process_execute icerisinde file_name olarak verilen parametre strtok_r ile file_name atama yaptım.*
- *Sonrasında ise process_execute thread_create (file_name ....) ile thread olusturuldu o sırada thread_create icerisinde olusturulan threain process olusturup thread icerisindeki process listesine ekledim.*
- *start_process icerisinde ise strtok_r ile file_name aldım ve char* token_ptr load fonksiyonuna parametre olarak verdim daha sonrasında setup_stack fonksiyonu üzerinde almıs oldugum arguman yapısını whitespace ile ayırıp tek tek malloc ile ayırdıgım kısma eklemek kaldı o sekılde tek tek for dongusunde strtok_r ile aldıgım null oluncaya kadar aldım ve malloc ayırdıgım yere ekledim.*

```
for (token = (char *) file_name; token != NULL;token = strtok_r (NULL, " ", token_ptr)) ;
```

- *Soldan baslayarak aldıgım ıcın ılkten baslayarak argumanlar [argv[0] ...) seklınde devam etti. bu ayırma ıslemıne ek olarak eger ki argc tanımladıgımız argv_size = ARGV_COUNTER (2) 'den buyukse o zaman reallocation yapıp ve argv_size da buyuttum ve memcopy yapıp eskı ekledıgım kısımları dırek memcopy yaptım ve bu sekılde eger buyuk gelmesı durumunda overflow onlendi.*

>> A3: Why does Pintos implement strtok_r() but not strtok()?

- *strtok() thread safe degıldır.*
- *strtok()_r ise thread safe dir.*

>> A4: In Pintos, the kernel separates commands into a executable name
>> and arguments. In Unix-like systems, the shell does this
>> separation. Identify at least two advantages of the Unix approach.

- *Unix sistemler daha komplex bir yapıya sahip.*
- *verilen executable ,arguman kısımlarının ayırılıp alt kısma gecis yapması gerekecektir.*

SYSTEM CALLS
==========

>> B1: Copy here the declaration of each new or changed `struct' or
>> `struct' member, global or static variable, `typedef', or
>> enumeration. Identify the purpose of each in 25 words or less.

syscall.h

```
#define NOT_LOADED 0       // ilk processin status degeri
#define SUCCESSFULLY_LOAD 1 // eger load fonksiyonu success olursa status degeri
#define LOAD_FAIL 2        // load fonksiyonu fail olursa status degeri
#define CLOSE_ALL -1       // file description type olarak define ettim.
#define ERROR -1           // diger durumlarda error kodu

struct process {
  int pid;    // process id
  int load;   // load islem statusu
  bool wait;  // process wait status true mu false mu.
  bool _exit; // exit process_exit..
  int status;
```

- *wait lock her bir child process te aynı sekılde bu durumu saglaması gerekır.*
- *aynı anda ulasma durumları bu sekılde kontrol altına alınacaktır.*

```
  struct lock lock_for_wait;
  struct list_elem elem;
};
```

- *process lerin dosya yapısı üzerinde islemlerini yapmaları icin*
- *bu kısımda islem yapılan file ,file description,*
- *list_elem (file icin gerekli element) listesi icerir.*

```
struct file_struct_for_process {
  struct file *file; // process 'in
  int fd;            // file description sayısı arttırılır
  struct list_elem elem;
};
```

- *verilen pid ile o anki threadin child process listesine eklenir.*
- *thread_create edildiginde o an child process eklenir.*
- *process_execute cagrıldıgında thread_create (thread) olusturulduktan sonra*
- *thread current'in child process listesine eklenir.*

```
struct process* add_process_current_thread (int pid);
```

- *verilen pid ile o anki threadin child listesinde var ise return edilir.*

```
struct process* child_process_check (int pid);
```

- *Verilen file descriptiona gore o anki thread teki file listesinde aynı fd ye ait olanlar kapatılır.*

```
void close_file_by_fd (int fd);
```

- *tum child process listesini silmek icin*

```
void remove_child_processes_all (void);
```

- *verilen child process o anki thread in child process listesinde aranır ve bulunması halinde silinir.*

```
void remove_process_by (struct process *cp);
```

syscall.c

- *maximum verilecek arguman sayısı system calll icin bu argumanlar uzerınden exit,filesize,ve diger syscall lar cagırılacak*
  #define MAX_ARGS 3
- *verilen frame uzerindeki sınırlarımız*
  #define VIRTUAL_ADRESSS_USER ((void *) 0x08048000)
  struct lock file_system_lock;
- *file system lock file system uzerinde islemler yapılırken*
- *lock yapmak icin kullanılacaktır.her bir file isleminde acquire edilir*
- *daha sonrasinda release edilir ve bu sayede baska threadler file system uzerinde islem yapacaktır.*

- *Process isleme alacagı file struct yapısını kendi icerisinde tuttugu listeye ekler.*
  int process_add_file (struct file *f);

- *verilen fd (file description ile file listesi icerisinde fd ye esit olan file yapısını return eder.*
  struct file* process_get_file (int fd);

- *Frame üzerinden argumanların alınmasını saglamaktayım.*
  void fetch_argumans_from_frame (struct intr_frame *f, int *arg, int n);

- *verilen addressin virtual address olarak frame sınırları icerisinde mi kontrol edilir.*
  void isvalid_frame_ptr (const void *vaddr);

- *Verilen adress lokasyonun verilen size boyunca uygun olmadıgını check ederiz (stack uzerinde,virtual address)*
  void boundry_check_buffer (void* buffer, unsigned size);

- *Verilen virtual address uzerınde stack icin tanımlanana adress aralıgında olup olmama durumunu ve stack uzerınden vırtual adress ıle ılıskılı olan kernel state adress degerını return eder.*

  int user_to_kernel_ptr(const void *vaddr);

>> B2: Describe how file descriptors are associated with open files.
>> Are file descriptors unique within the entire OS or just within a
>> single process?

- *Her bir process uzerinde file eklenmesi durumunda o anki file icin tasarladıgım fd yi o anki thread in fd sine set ettim daha sonrasında yeni bir file gelmesi durumunda file_struct_for_process deki file description o anki threadin fd degerine esitledim. daha sonrasinda o anki threadin fd 'sini bir arttırdım.(ilk file icin fd = 0 olsun , ikincisi icin fd = 1 ... bu sekilde devam eder)*

- *Her bir file icin bir fd durumu söz konusu olucak sekilde tasarladım*

- *Her bir process icin file_struct_for_process yapısı uzerinde file ve file description yapısı olusturdum.Bu sekılde her bir process için her bir file isleminde fd yi o anki thread uzerinde tuttugum fd (file description) üzerinden set ettim.*

- *Bazı durumlarda yani fonksiyonlarda (process_get_file fonksiyonu) verilen file description ile thread.h üzerinde tuttugum file listesi üzerinde (o anki calısan threadin file listesi uzerinde) arama yaparak egerki verilen file descriptiona esitse o anda file descriptionnın tuttugu file return*

ettim.Herhangi bir sekilde liste içerisinde kapatılması gereken file 'ı da aynı sekilde file_listesi (o anki thread) üzerinde arayarak eger liste uzerindeki fd kapatılması gereken file 'ı tutuyorsa kapatılmasını sagladım.

- *thread.h 'da tanımlandıgım fd file_liste uzerinde bulunan en son eklenen file description sayısını tutmaktadır.(process_add_file)Bu sayede o anki calısan thread uzerinde bulunan process'e file eklenmesi durumunda fd (thread_current() -> fd ++) yi bir arttırdım.Ve thread uzerınde bulunan file_list üzerine ekledim.Bu sekılde file listesindeki file ları file description ile indexledim.*

>> B3: Describe your code for reading and writing user data from the
>> kernel.

- *Data okunması durumunda virtual address lokasyonunda sınırları kontrol ederek her bir read yada write isleminde (system call) oncelıkle verilen file description valid olup olmama durumuna baktım.*

- *Egerki file description SDTOUT_FILENO ise putbuffer ile console uzerine bastım.Eger ki STDIN_FILENO ile file description olarak karsılasırsam o zamanda input_getc ile character character olarak aldım.Ayrıca STDIN_FILENO ve STDOUT_FILENO olmama durumunda o anki thread uzerinde bulunan file listesinde arama yaptım eger varsa tamam o zaman o file uzerinde read / write islemlerini yaptım.Eger yoksa lock_acquire yaptıgım file_system_lock geri release ettim.*

- *Ayrıca sistem uzerinde user virtual address'in valid olma durumunu kontrol etmek gerekiyor Bunun icinde isvalid_frame_ptr fonksiyonu ile kontrol ettim.*

>> B4: Suppose a system call causes a full page (4,096 bytes) of data
>> to be copied from user space into the kernel. What is the least
>> and the greatest possible number of inspections of the page table
>> (e.g. calls to pagedir_get_page()) that might result? What about
>> for a system call that only copies 2 bytes of data? Is there room
>> for improvement in these numbers, and how much?

- *Her bir system call üzerinde (exec ,remove ,create ,wait ..) stack uzerinde virtual address ile user_to_kernel_ptr fonksiyonu ile*
- *pagedir_get_page fonksiyonu uzerinden kernel state memory adress return ettim. (user_to_kernel_ptr)*

>> B5: Briefly describe your implementation of the "wait" system call
>> and how it interacts with process termination.

- *wait fonksiyonu kendi icerisinde process_wait fonksiyonunu cagırmaktadır. process_wait uzerinde verilen pid ye gore child list uzerinde bulunuyorsa wait statementı true ise return ERROR eger wait halinde degilse process'in wait = true yaptım ve daha sonrasında beklemesı icin disarıdan bır etkı olmadıgı surece*
- *while(!process -> exit) cıkıs olmadıgı surece barrier ile sistem üzerinde process bekletilir.*
- *Eger ki o anki thread icerisinde bulunmuyorsa ERROR return edilir.*

>> B6: Any access to user program memory at a user-specified address
>> can fail due to a bad pointer value. Such accesses must cause the
>> process to be terminated. System calls are fraught with such
>> accesses, e.g. a "write" system call requires reading the system
>> call number from the user stack, then each of the call's three
>> arguments, then an arbitrary amount of user memory, and any of
>> these can fail at any point. This poses a design and
>> error-handling problem: how do you best avoid obscuring the primary
>> function of code in a morass of error-handling? Furthermore, when
>> an error is detected, how do you ensure that all temporarily
>> allocated resources (locks, buffers, etc.) are freed? In a few
>> paragraphs, describe the strategy or strategies you adopted for
>> managing these issues. Give an example.

- *System call icerisinde bad pointer durumu istenilmeyen bir state (adress lokasyonuna) erisim soz konusu oldugunda olusucaktır.is_valid_ptr fonksiyonu ile erisilmek istenen adress lokasyonlarını kontrol ediyorum.Eger ki hata olma durumunda ERROR olarak tanımladıgımız deger ıle return edicek en son exception olma durumunda exit(-1) ile statu printf edilerek thread_exit yaptım.*

- *Her file system uzerinde islem yapmaya calıstıgımızda ise lock_acquire ile file system lock ettim daha sonrasında file system uzerinde işlemlerimi yapıp geri lock_release ettim.bu sekılde aynı anda ulasma durumunu kontrol altına almıs oldum.Buffer durumlarında verilen argumanlara gore arguman size olarak default tanımlanan degerden buyuk olursa reallocation ile yeni yer alıp eski kısımları memcpy yapmaktayım bu sayede hata durumunuda kontrol etmis oluyorum.*

- *Arguman eklenmesi durumunda bazı durumlarda stack uzerine ekleme olmama durumunda LOADING_FAILED gibi tanımlamalar ile bu durumdan haberdar oldum ve thread_exit ile islem sonlandırılmıs oldu.*

>> B7: The "exec" system call returns -1 if loading the new executable
>> fails, so it cannot return before the new executable has completed
>> loading. How does your code ensure this? How is the load
>> success/failure status passed back to the thread that calls "exec"?

- *Her bir process kendi icerisinde load ve exit boolean olarak yapı tutar bu sayede olusturulan processlerin hangi state te bulundugu hakkında bilgimiz olucaktır.*

- *Eger herhangi bir yerden bir interrupt gelmesi durumunda exit olması halide kontrol edilmeli bundan dolayı process icerisinde exit (bool) yapısını olusturdum.*

- *oncelikle exe fonksiyonumuz process_execute ediyor daha sonrasında verilen command line baglı olarak olusturulan process pid'sini return eder eger bir hata durumu soz konusu ıse check ıslemı yaparız (child list icerisinde) daha sonrasında eger o child load success olmusssa tamamdır islemlerimiz dogru fakat hata olma durumunda yanı LOADING_FAILED olmussa error ile return ederim ,yada ilk process initialize edilme durumunda takılı kalmıssa bu durumda process olusumunda sıkıntı cıkmıstır.Eger hersey dogru sekılde tamamlanmıssa process_execute edilen process pid si return edilir.*

>> B8: Consider parent process P with child process C. How do you
>> ensure proper synchronization and avoid race conditions when P
>> calls wait(C) before C exits? After C exits? How do you ensure

>> that all resources are freed in each case? How about when P
>> terminates without waiting, before C exits? After C exits? Are
>> there any special cases?

- *Bu durum ıcın her bir child processin exit state 'i ne durumda oldugu kontrol edilerek isleme baslarım cunki yukarıda B7 dedigim gibi herhangi bir yerden*
- *exit durumu olusabılır bu durumda basta ılk olarak wait edilmek istenen child process liste icerisinde varlıgı kontrol edilir child_process_check().Daha sonrasında*
- *eger liste icerisinde degılse return ERROR eger liste icerisinde fakat process -> exit = TRUE ise return ERROR state ile return ederim.çünkü process onceden exit durumunda bunun ıcın wait etmeden once bu kontrolu gerceklestırmem gerekir.*

- *process exit olurken tum acmıs oldugu file lar close_file_by_fd ile close edilir daha sonrasında tum thread uzerindeki thread listesindeki bulunanlar remove edilir.*

>> B9: Why did you choose to implement access to user memory from the
>> kernel in the way that you did?

- *Kernel state uzerınden user state ulasım ıcın user_virtualaddress_to_kernel_ptr fonksiyonunu kullandım,*
- *oncelıkle verilen adressin valid olma (PHYS_BASE ile VIRTUAL_ADRESSS_USER arasında olma durumu) durumunu kontrol edip daha sonrasında pagedir_get_page fonksiyonundan return edilen adressi return ettim.*
- *Ayrıca her bir system call uzerinde verilen adress 'in uygun olup olmama durumunu ve ona gore fetch edılen argumanların adreslerinin kernel state uzerınde user_virtualaddress_to_kernel_ptr return edilen adressine uygun bir sekilde systemcall yaptım.*

>> B10: What advantages or disadvantages can you see to your design
>> for file descriptors?

- *Avantajı file description liste uzerindeki fileların indexli bir sekılde tutulmasını saglıyor bu sayede ben ayarlamıs oldugum fd lere gore liste uzerinde arama ile liste uzerinde o file 'in var olup olmama durumunu kontrol edebiliyorum.*

>> B11: The default tid_t to pid_t mapping is the identity mapping.
>> If you changed it, what advantages are there to your approach?

- *tid_t yi pid_t ile mapped etmeksem o zaman multiple threads destegi olusur fakat pintos tarafından multiple thread desteklenmemektedir.*