LINKÖPINGS UNIVERSITET

TDIU16: Process- och operativsystemprogrammering

Accesskontroll

Klas Arvidsson, Daniel Thorén, Filip Strömbäck

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2018

2020-03-06

1 Bakgrund

Alla program i Pintos (och i andra operativsystem) körs i en nedlåst miljö som ofta kallas för usermode. Operativsystemet ser till så att processer i usermode inte får göra mycket med än att läsa från och skriva till det minne som är reserverat specifikt för processen. Däremot kan processen med hjälp av systemanrop be operativsystemet att göra saker som processen normalt inte får göra. Eftersom operativsystemet inte körs i usermode får det göra vad som helst med systemet, och därmed är det viktigt att operativsystemet är mycket noga med att kontrollera så att programmet i usermode inte försöker göra något konstigt.

I den här uppgiften fokuserar vi på fallet när programmet i usermode ber operativsystemet att läsa från eller skriva till minne (exempelvis om programmet ville läsa från en fil). Från operativsystemets synvinkel är då två saker viktiga att kontrollera:

1. Att den adress vi försöker komma åt faktiskt finns tillgänglig i RAM. Annars kommer ett pagefault att genereras, och hela operativsystemet kraschar.
2. Att den process som har anropat systemanropet skulle få komma åt adressen i normala fall. Annars skulle det exempelvis vara möjligt att läsa känslig information som bara operativsystemet har tillgång till i normala fall, eller skriva över viktiga datastrukturer som gör att operativsystemet kraschar eller på annat sätt slutar fungera.

Mer information om hur detta fungerar finns i Pintos-Wiki under rubriken ”Minne”, se särskilt underrubriken ”Accesskontroll”.

2 Mål

Målet med uppgiften är således att implementera funktionalitet som kan bedömma om minnesåtkomster är okej att utföra utan risk att krascha operativsystemet. Vi kommer att flytta denna koden till Pintos sedan.

3 Uppgift

Du skall skriva logiken som verifierar att allt minne inom ett visst intervall är giltigt. Till din hjälp har du ett antal funktioner och konstanten *PGSIZE* (se header-filen):

```
void *pg_round_down(const void* adr);
```

Returnerar första adressen i samma sida som *adr*.

```
unsigned pg_no(const void* adr);
```

Returnerar numret på sidan som innehåller *adr*. Sidnummer börjar räkna på 0, som allt annat i C.

```
void *pagedir_get_page (void *pd, const void *adr);
```

Använder översättningstabellen *pd* för att slå upp fysisk adress motsvarande *adr*. Om översättningen misslyckas returneras *NULL*. I denna uppgift används ett simulerat testsystem där *pd* kan anges till *NULL*.

Notera att denna funktion är relativt dyr. Du vill alltså se till att inte anropa den fler gånger än vad du faktiskt behöver! I testprogrammet tar varje anrop till `pagedir_get_page` några hundra millisekunder, så om testerna tar lång tid att köra anropar du antagligen funktionen för många gånger!

```
bool is_end_of_string(char* adr);
```

Returnerar *true* om adressen *adr* innehåller ett noll-tecken, `'\0'`. Eftersom koden endast simulerar systemet går inga adresser att läsa eller skriva. Därför måste du använda denna funktion för att avgöra om en sträng är slut. Funktionen ersätter alltså testet `*adr == '\0'`.

De funktioner du måste implementera är:

```
bool verify_fix_length(void* start, int length);
```

Kontrollera alla adresser från och med *start* till och *inte* med *(start+length)*. Returnerar *true* om alla adresser i intervallet är giltiga.

```
bool verify_variable_length(char* start);
```

Kontrollera alla adresser från och med *start* till och med den adress som först innehåller ett noll-tecken, `\0`. (C-strängar lagras på detta sätt.) Returnerar *true* om alla adresser som används av strängen är giltiga.

4 Given kod

Ett givet testprogram, filen *verify_adr.c* samt *pagedir.h* och *pagedir.o*, finns tillgängliga i mappen `src/standalone/lab14`. Tänk på att dessa filer inte implementerar ett verkligt system. Adresserna går därför inte att läsa eller skriva, och det behöver du inte heller eftersom du endast skall kontrollera om de är giltiga. Då du behöver testa om en adress innehåller ett noll-tecken (för att hitta slutet på en sträng) finns en funktion som simulerar just den läsningen given ovan. Testprogrammet skall inte skriva ut några fel, och bör avsluta inom en minut när du gjort rätt (gör du fel kommer det ta avsevärt längre tid eftersom varje uppslagning är mycket långsam i det simulerade systemet).

För att kompilera koden använder du som vanligt *gcc* med alla käll- och objektfiler som skall ingå:

```
gcc -Wall -Wextra -std=gnu99 -m32 -g verify_adr.c pagedir.o
```