Systemy operacyjne

Lista zadań nr 0

Na zajęcia 8 października 2020

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Operating Systems: Three Easy Pieces – Introduction[1]
- Computer Systems: A Programmer's Perspective (3rd edition) – 7.8, 7.9, 8.1, 9.7

UWAGA! W trakcie prezentacji należy zdefiniować pojęcia oznaczone **wytłuszczoną** czcionką.

Zadanie 1. Opisz różnice między **przerwaniem sprzętowym** (ang. *hardware interrupt*), **wyjątkiem procesora** (ang. *exception*) i **pułapką** (ang. *trap*). Dla każdego z nich podaj co najmniej trzy przykłady zdarzeń, które je wyzwalają. W jakim scenariuszu wyjątek procesora nie oznacza błędu czasu wykonania programu? Kiedy pułapka jest generowana w wyniku prawidłowej pracy programu?

Zadanie 2. Opisz mechanizm **obsługi przerwań** bazujący na **wektorze przerwań** (ang. *interrupt vector table*). Co robi procesor przed pobraniem pierwszej instrukcji **procedury obsługi przerwania** (ang. *interrupt handler*) i po natrafieniu na instrukcję powrotu z przerwania? Czemu procedura obsługi przerwania powinna być wykonywana w **trybie jądra** (ang. *kernel mode*) i używać stosu odrębnego od stosu użytkownika?

Zadanie 3. Bazując na formacie `ELF` (ang. *Executable and Linkable Format*) opisz składowe pliku wykonywalnego. Czym różni się **sekcja** od **segmentu**? Co opisują **nagłówki programu**? Skąd system operacyjny wie, pod jakim adresem ma umieścić segmenty programu i gdzie położona jest pierwsza instrukcja programu?

Wskazówka: Skorzystaj z narzędzia «`readelf`».

Zadanie 4. Zapoznaj się z rozdziałami 3.4 i $\mathrm{A}.2$ dokumentu System V Application Binary Interface AMD64 Architecture Processor Supplement[2] i odpowiedz na następujące pytania:

- W jaki sposób jądro musi przygotować **przestrzeń adresową** procesu? Co musi znajdować się na stosie w momencie wywołania procedury «`_start`»? Do czego służy $\mathrm{auxiliary\ vector}$? Można go wyświetlić wydając przykładowe polecenie «`LD_SHOW_AUXV=1 /bin/true`».
- W jaki sposób wywołać funkcję jądra? W których rejestrach należy umieścić argumenty? Gdzie można spodziewać się wyników i jak jądro sygnalizuje niepowodzenie **wywołania systemowego**?

Zadanie 5. Przypomnij jak wygląda mechanizm **tłumaczenia adresów** bazujący na wielopoziomowej tablicy stron procesorów z rodziny `x86-64`. Przedstaw algorytm obliczania **adresu fizycznego** na podstawie **adresu wirtualnego** z uwzględnieniem uprawnień dostępu. Jaką rolę w procesie tłumaczenia odgrywa **pamięć TLB**?

Ściągnij ze strony przedmiotu archiwum «`so20_lista_0.tar.gz`», następnie rozpakuj i skompiluj źródła poleceniem «`make`».

Zadanie 6. Uruchom program «`1_ls`» pod kontrolą narzędzia «`ltrace -S`». Na podstawie śladu wykonania programu zidentyfikuj, które z **wywołań systemowych** są używane przez procedury: «`opendir`», «`readdir`», «`printf`» i «`closedir`». Do czego służy wywołanie systemowe «`brk`»? Używając debuggera «`gdb`» i polecenia «`catch syscall brk`» zidentyfikuj, która funkcja używa «`brk`».

Zadanie 7. Pod kontrolą narzędzia «`strace`» uruchom program «`2_cat`» korzystający bezpośrednio z wywołań systemowych do interakcji ze **standardowym wejściem i wyjściem**. Pokaż, że program oczekuje na odczyt na deskryptorze pliku 0 i pisze do pliku o deskryptorze 1. Naciśnij kombinację klawiszy «`CTRL+D`» kończąc wejściowy strumień danych – co zwróciło «`read`»? Zmodyfikuj program tak, by czytał z pliku podanego w linii poleceń. Co się stanie, jeśli przekażesz **ścieżkę** do katalogu zamiast do pliku regularnego?

[1] `http://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf`
[2] `https://www.uclibc.org/docs/psABI-x86_64.pdf`