Przedmiot WSO 1
Pytania egzaminacyjne podstawowe

1. Co to jest system operacyjny?

System operacyjny spełnia 2 główne zadania:
1.Podział zasobów – SO musi rozdzielać zasoby (procesor , pamięć i urządzenia we-wy) komputera między wielu równoczesnych jego użytkowników.
2. Tworzenie maszyny wirtualnej – przekształcenie „surowego" sprzętu w maszynę, która jest łatwiejsza w użyciu – maszynę wirtualna, której właściwości użytkowe są inne niż danej maszyny fizycznej ale znacznie przydatniejsze dla użytkownika.

2. Co oznacza wielodostępność, a co wielozadaniowość?

Unix jest wielozadaniowym, wielodostępnym i interaktywnym systemem operacyjnym o szerokim zakresie zastosowań.

Wielozadaniowość oznacza, że system może wykonywać jednocześnie wiele zadań (programów), w tym również na rzecz jednego użytkownika.

Wielodostępność oznacza, że z usług systemu może korzystać jednocześnie wielu użytkowników.

3. Jakie funkcje spełnia wielodostępny system operacyjny i jakie podsystemy wchodzą w jego skład?

System operacyjny Linux jest systemem wielodostępnym i wielozadaniowym, tak więc jądro systemu operacyjnego musi realizować następujące zadania:

- dzielić zasoby,
- szeregować procesy, koordynować ich pracę, zarządzać nimi, umożliwiać ich współpracę
- umożliwiać współpracę z urządzeniami peryferyjnymi (wspólnymi dla wielu użytkowników),
- zapewnić bezkolizyjne wykorzystanie pamięci operacyjnej,
- zorganizowć system plików, zarządzać plikami, chronić pliki,
- obsługiwać błędy

W systemach wielodostępnych (lub interaktywnych) użytkownik może wykonywać jeden lub wiele programów wywołując je za pośrednictwem terminala i mioże posługiwać się terminalem w celu nadzorowania przebiegu programów i sterowania ich wykonywaniem. SO rozdziela zasoby obliczeniowe między różnych użytkowników w taki sposób , aby wydawało się iż każdy z nich ma wyłączny dostęp do całej maszyny.

4. Co jest przedmiotem standaryzacji w wielodostępnych systemach operacyjnych?

5. Co to jest proces?

UNIX pozwala swoim użytkownikom rozpoczynać różne zadania jako procesy współbieżne. Proces jest to (w uproszczeniu) aktywny program, któremu przydzielono pewien obszar pamięci i pewne zasoby systemowe. Użytkownik poprzez powłokę lub z własnych programów może rozpoczynać, synchronizować i kończyć procesy. Zaletami mechanizmu procesów jest to, że:

- kompleksowe zadania mogą zostać rozłożone na autonomiczne części, które potrzebują każdorazowo tylko pewnej części zasobów,
- czasochłonne działania mogą być wykonywane drugoplanowo (np. wsadowo), a użytkownik może w tym czasie przeprowadzać inne działania.

Oprócz tego mechanizm procesu stwarza doskonałą podstawę do realizacji takich działań administracyjnych jak zarządzanie kolejkami, usługi oparte o kalendarz, regularne wykonywanie kopii systemu i danych itd.

Po otwarciu sesji (komendą **login**) rozpoczynany jest dla każdego użytkownika jego pierwszy proces, który pozostaje aktywny na czas sesji. Ten proces powłoki (Login Shell) inicjalizuje dla każdej komendy dalsze procesy, na których wykonanie powłoka czeka (procesy synchroniczne), lub które przebiegają równolegle drugoplanowo (procesy synchroniczne).

Nowy proces jest tworzony i rozpoczynany przez proces aktywny. Synchronizacja procesów jak i ich komunikacja może odbywać się poprzez pliki, sygnały, semafory, wspólną pamięć i wiadomości. Programiści mają dostęp do tych mechanizmów poprzez odpowiednie funkcje systemowe. Język powłoki stwarza również szereg możliwości efektywnego wykorzystania koncepcji procesu

Program a proces

Zacznijmy od określenia co to jest program. Jest to zestaw instrukcji i danych przechowywanych na dysku w postaci wykonywalnego pliku zwykłego. W i-węźle takiego pliku znajduje się informacja o tym, że plik jest wykonywalny. Po to, by uruchomić program jądro systemu musi utworzyć nowy proces. Każdy proces składa się z trzech segmentów:

- segment instrukcji
- segment danych użytkownika
- segment danych systemowych

Te 3 segmenty nazywane są również środowiskiem, w którym program się wykonuje. Po zainicjowaniu procesu nie ma już powiązania między procesem a programem przez ten proces wykonywanym. Można na podstawie jednego programu zainicjować kilka procesów wykonywanych jednocześnie i również nie ma powiązania między takimi procesami. Jądro systemu może wszystkim procesom zainicjowanym na podstawie tego samego programu przydzielić ten sam segment instrukji - cel takiego postępowania jest oczywisty, mianowicie oszczędzanie pamięci.

W segmencie danych systemowych są między innymi informacje o bieżącym katalogu, o otwartych plikach, o czasie wykorzystania procesora i wiele innych, których część poznamy teraz. Proces nie może w sposób bezpośredni zmieniać danych systemowych.

Każdy proces (z wyjątkiem procesu init) jest tworzony przez jądro systemu na "prośbę" bieżąco wykonywanego procesu, który staje się przodkiem nowego procesu - potomka. Potomek dziedziczy część danych systemowych procesu macierzystego (np. jeśli proces macierzysty miał otwarte pliki, to potomek ma je również otwarte).

Każdemu procesowi przyporządkowany jest identyfikator procesu - PID. Jest on liczbą całkowitą dodatnią i jego wartość jednoznacznie określa proces. Każdy proces ma swój proces macierzysty, a jego identyfikator określony jest skrótem PPID (parent proces ID). Można jedno zadanie podzielić na kilka procesów i utworzyć grupę procesów. Grupie procesów przyporządkowywany jest również identyfikator PGID.

Rozpoczęcie sesji przez użytkownika wiąże się z pobraniem jego UID i GID i uczynienie ich identyfikatorami odpowiednio użytkownika i grupy użytkowników dla pierwszego uruchamianego procesu - jest nim powłoka (np. bash). Po takim przypisaniu otrzymujemy rzeczywity identyfikator użytkownika RUID, oraz grupy RGID. Z każdym procesem związane są jeszcze dwa identyfikatory, są to obowiązujący (również nazywany efektywnym) identyfikator użytkownika - EUID i grupy EGID. W większości zadań rzeczywiste

identyfikatory użytkownika i grupy są takie same jak identyfikatory obowiązujące. Programy, które muszą operować na plikach niedostępnych zwykle dla użytkownika, ustawiają odpowiednio efektywny identyfikator użytkownika. Takim programem jest np. program zmieniający domyślną powłokę użytkownika. Identyfikator obowiązujący dla danego procesu służy do określania praw dostępu. Np. program, który jest własnością root'a i ma ustawiony specjalny bit dostępu, zwany bitem ustanowienia identyfikatora użytkownika (set user ID) powoduje, że użytkownik na czas wykonania tego programu otrzymuje uprawnienia właściciela programu - root'a. Po wykonaniu polecenia

- l /usr/bin/chsh

otrzymasz

```
-rws--x--x   1 root     root        13800 Apr 17  1999 /usr/bin/chsh
```

w którym prawo dostępu na wykonanie dla właściciela zostało ustawione na s, a to oznacza, że każdy użytkownik mający prawo wykonania tego pliku dostaje prawo root'a (tylko na czas wykonywania tego programu).

Każdy proces ma swój priorytet (nice), służy on do szeregowania procesów przez jądro systemu. Proces użytkownika może obniżyć swój priorytet, jedynie proces super-użytkownika może podwyższyć swój priorytet.

Proces jest to pojedynczy program wykonujący się we własnej wirtualnej przestrzeni adresowej . Różnica między procesem , a zadaniem czy poleceniem polega na tym , że te ostatnie składać się mogą z wielu procesów realizujących określoną czynność . Proste polecenie takie jak np. ls wykonuje się jako jeden proces. Wykonywanie bardziej złożonych poleceń , korzystających z potoków , powoduje uruchamianie jednego procesu dla każdego segmentu potoku.

Slajdy – wso_2_28

6. Co to jest blok kontrolny procesu i do czego służy?

Każdy proces można reprezentować przez deskryptor procesu , nazywany też czasem blokiem kontrolnym procesu (ang .process control block) lub wektorem stanu , który jest obszarem pamięci operacyjnej zawierającym wszystkie ważne informacje o danym procesie.

Slajdy – wso_2_29

7. Co oznacza współbieżne wykonywanie procesów?

Współbieżność można rozumieć jako uaktywnienie kilku procesów (tj . wykonywanie kilku programów) w tym samym czasie. Jeżeli procesów jest więcej niż procesorów to można uzyskać pozorną współbieżność przełączając procesor od jednych procesów do drugich. Jeżeli przełączenia dokonywane są w dostatecznie małych okresach czasu , to system obserwowany przez dłuższy czas pozwoli stworzyć iluzje współbieżności. Pozorna współbieżność powstaje w wyniku przeplatania procesów w jednych procesorze.

Slajdy – wso_2_28

8. Jak powstaje nowy proces?

Nowy proces w systemie uruchamiany jest w następujący sposób. Po pierwsze istniejący proces wykonuje czynność zwana rozwidleniem (forking) , polegającą na utworzeniu dokładnej kopji samego siebie. Utwoerzony w ten sposób nowy proces określanuy jako proces dziecko, posiada własny identyfikator PID , chociaż jego srodowisko jest dokładnie

takie samo jak procesu rodzica. W tym momencie w wyniku wykonania funkcji systemowej o nazwie exec treść programu zawarta w przestrzeni adresowej procesu dziecka ulega wymianie na treśc tego programu , który ma być wykonany w ramach danego procesu (fork –and – exec) Treść nowego programu kompletnie zastępuje tę odziedziczoną od procesu rodzica. Bez zmian pozostaje jesdnak srodowisko procesu , w tym ustawienia zmiennych , przypisania strumienia standardowergo wejscia, stand wyj oraz stan wyj błędów, a także priorytet procesu.

Slajdy – wso_2_30

9. Jak proces jest kończony?

W momencie kiedy proces kończy swoje działanie , proces rodzic zostaje o tym poinformowany przez odpowiedni sygnał . c.d. w następnym odcinku ;)

Slajdy – wso_2_31

10. Jak działa funkcja systemowa exec?

Przeznacznie:

Funkcja `exec()` zmienia kontekst poziomu użytkownika danego procesu na kopie programu wykonywalnego umieszczonego w pliku dyskowym. Jeżeli funkcja zostanie wykonana bezbłędnie, to poprzedni kontekst procesu znika i już nigdy nie zostanie odtworzony. Warto zwrócić uwagę, że funkcja exec nie powoduje powstania nowego procesu, lecz jedynie dokonuje wymiany kontekstu istniejącego procesu (wywołującego tę funkcję), który otrzymuje nowy segment danych, kodu i stosu.

Przykładowe zastosowanie:

W połączeniu z funkcjami systemowymi `fork` i `wait` można za pomocą funkcji `exec` zrealizować następujący mechanizm: przekazujemy sterowanie do uruchamianego programu, a po jego zakończeniu, kontynuujemy wykonywanie pierwotnego procesu wywołującego bez utraty żadnych danych.

Po wykonaniu funkcji `fork()` jądro tworzy - w przypadku sukcesu - nowy proces - bliźniaka. Aby nowopowstały proces wykonał kod leżacy w zewnetrznym pliku wykonywalnym należy wywołaż funkcje systemowa z rodziny `exec()`.

11. Jak działa funkcja systemowa fork?

Funkcja `fork()` - tworzenie nowego procesu
Zadaniem funkcji fork() jest tworzenie nowego procesu, będącego potomkiem procesu wywołującego. Wszystkie procesy w systemie powstają za jej pomocą, oprócz procesu o pidzie 0, który jest tworzony wewnętrznie przez jądro przy ładowaniu systemu do pamięci.

Podstawowy interfejs programisty:

```
pid_t fork(),
```

gdzie wartość zwracana jest liczbą określającą tożsamość procesu:

- ojciec otrzymuje pid potomka
- potomek otrzymuje zero

lub informującą o błędzie w przypadku niepowodzenia przy tworzeniu nowego procesu: -1. Stąd typowy sposób użycia fork:

```
main(argv,argc)
```

```
        {
          switch(fork())
          {
            case -1 : /* zgłoszenie błedu */
            case  0 : /* bezpośrednio kod potomka lub wywołanie funkcji
exec */
            default : /* kod ojca */
          }
        }
```

Proces potomny dziedziczy po macierzystym cały jego kontekst, z jedynym wyjątkiem - właśnie wartością zwracaną przez `fork()`. Powstanie nowego procesu jest niemożliwe w przypadku braku pamięci lub przy przekroczeniu limitu na liczbę procesów.

fork

Najważniejsze czynności wykonywane przez funkcję `fork()`:

- utworzenie nowej struktury `task_struct` dla procesu,
- utworzenie nowego stanu kontekstu jadra,
- przydzielenie nowego identyfikatora procesu,
- kopiowanie zasobow procesu macierzystego,
- utworzenie nowej struktury `mm_struct`,
- utworzenie nowego katalogu stron, lub podłączenie do katalogu procesu macierzystego,
- "kopiowanie" pamięci wirtualnej.

Cztery ostatnie pozycje dotyczą pamięci.

12. Co to są funkcje systemowe?

13. W jakich stanach może być proces?

W strukturze `task_struct` umieszczone jest pole

```
volatile long state;
```

Pole to określa stan, w jakim aktualnie znajduje się dany proces. Proces może, reagując na pewne zdarzenia i okoliczności, zmieniać swój stan. Pole to może przyjąć jedną z następujących wartości:

- **TASK_RUNNING** (0) - proces jest procesem, który aktualnie wykorzystuje procesor lub czeka na jego przydzielenie (jest na liście procesów gotowych do wykonywania).
- **TASK_INTERRUPTIBLE** (1) - proces czeka na określone zdarzenie. Różnica pomiędzy tym stanem, a TASK_UNINTERRUPTIBLE jest taka, że tutaj proces obsługuje nadchodzące sygnały. Proces w tym stanie jest po prostu uśpiony i po nadejściu sygnału lub zajściu zdarzenia, na które czekał, umieszczany jest w kolejce procesów gotowych do wykonania.
- **TASK_UNINTERRUPTIBLE** (2) - podobnie jak TASK_INTERRUPTIBLE, ale w tym stanie proces nie obsługuje nadchodzących sygnałów. Proces znajduje się w tym stanie zwykle w oczekiwaniu na spełnienie jakiegoś warunku związanego ze sprzętem.
- **TASK_STOPPED** (4) - proces został wstrzymany po otrzymaniu odpowiedniego sygnału (**SIGSTOP, SIGSTP, SIGTTIN, SIGTTOU**) lub będąc monitorowanym przez inny proces przy użyciu wywołania systemowego `ptrace` przekazał sterowanie do procesu monitorującego. Opuszczenie tego stanu możliwe jest jedynie po otrzymaniu sygnału **SIGCONT**.

- **TASK_ZOMBIE** (8) - proces został przerwany (skończył swój komputerowy żywot), ale jego struktura znajduje się jeszcze w tabeli procesów.
- **TASK_SWAPPING** (16) - stała ta jeszcze nie jest używana w wersji systemu 2.2. Żaden proces nie znajduje się nigdy w tym stanie.

O procesie, który znajduje się aktualnie w stanie **TASK_RUNNABLE** i jest tym, który ma aktualnie przydzielony procesor, mówimy, że się właśnie wykonuje. Proces taki może wykonywać się w jednym z dwóch następujących trybów:

- **Tryb użytkownika** - tryb podstawowy procesu.
- **Tryb jądra** - w tym trybie możliwe jest wywoływanie funkcji systemowych.

Chyba nie oto chodziło , co? Szczególnie że dotyczy to raczej Linuxa niż Unixa.
Może więc to:

Procesy mogą znajdować się w kilku stanach

- tworzenie procesu (SIDL)
- wykonywanie procesu (SRUN)
- zatrzymany proces (SSTOP)
- oczekiwanie na zdarzenie (SSLEEP)
- usuwanie procesu (SZOMB)

Proces ze stanu SIDL, jeśli uzyska wystarczające zasoby przechodzi do stanu SRUN. Od tego momentu stan procesu oscyluje między SRUN, SSLEEP, SSTOP do czasu zakończenia tzn. przejścia w stan SZOMB. Proces znajduje się w stanie SZOMB do chwili powiadomienia procesu macierzystego o zakończeniu procesu.

Wiemy już, że każdy proces (z wyjątkiem procesu init) jest powoływany do życia przez inny proces. Do powoływania do życia nowych procesów służą funkcje systemowe: grupa funkcji exec, oraz funkcja fork. Funkcje z grupy exec ponownie inicjują ten sam proces na podstawie wskazanego programu (podmieniają segment instrukcji) - proces zostaje ten sam. Funkcja fork, tworzy nowy proces przez skopiowanie wszystkich segmentów procesu, nowy proces nie jest inicjowany na podstawie programu. Użyteczność funkcji z grupy exec byłaby dość ograniczona, zaś funkcji fork bez exec praktycznie żadna. Dopiero łączne użycie tych funkcji daje potężne narzędzie programistyczne.

Jeśli z poziomu powłoki piszemy dowolne polecenie, to najpierw jest wywoływana funkcja fork, tworząca nowy proces (jest to kopia wszystkich segmentów procesu bash, a następnie odpowiednia funkcja z grupy exec, która podmieni segment instrukcji na podstawie wydanego polecenia. Z poziomu bash można wydać polecenie exec, które za pośrednictwem funkcji systemowej z grupy exec pozwoli na zastąpienie segmentu instrukcji procesu bash, nowym segmentem instrukcji. Wtedy po zakończeniu działania procesu nie należy się spodziewać, że proces bash będzie istniał. Można spróbować w jednym oknie napisać (najpierw sprawdzić PID procesu bash):

- exec sleep 100

w drugim oknie sprawdzić PID procesu sleep i poczekać na zakończenie jego działania - okno, w którym uruchomione było polecenie exec zniknie, ponieważ proces bash został zastąpiony procesem sleep, a ten skończył działanie.

Stan procesu może przyjmować jedną z głównych wartości , proces może być :
1. wykonywany (running) bieżąco przez jakiś procesor

2. wykonywalny (runnable) czylio mógłby być wykonywany gdyby przydzielono do niego jakiś procesor
3. niewykonywalny (unrunnable), nie mógłby użyć procesora nawet gdyby mu go przydzielono. Najczęstszą przyczyną niewykonywalności jest fakt , że oczekuje on na zakończenie przesyłania danych z urzadzenia zewn .

Slajdy - wso_2_29

14. Co to jest "shell"?

Po rozpoczęciu sesji zaczynasz pracę w swojej domowej kartotece i jednocześnie uruchamiasz program ogólnie zwany powłoką (po angiesku shell czyli powłoka, skorupka). Domyślnie jest to powłoka o nazwie bash. Wszystko co napiszesz na klawiaturze po naciśnięciu znaku [ENTER] zostanie zinterpretowane przez program bash. Zgodnie z nazwą powłoka tworzy zewnętrzną warstę systemu operacyjnego, warstwę z którą kontaktuje się użytkownik systemu. Na rys 1 jest przedstawiona w sposób schematyczny struktura warstwowa systemu. Powłoka jest:

- programem
- językiem programowania (można pisać programy w języku powłoki, programy te nazywane są skryptami)
- interpreterem (programy pisane w języku powłoki nie są kompilowane, a interpretowane)
- interface'm użytkownika

Slajdy - wso_2_20

15. Podać przykłady programów shell i ich właściwości?

16. W jaki sposób program shell interpretuje polecenie?

Shell jest zawsze gotowy do przyjęcia nowego polecenia. Wprowadzenie ciągu znaków zakończonych naciśnięciem klawisza Enter jest sygnałem do pobrania wprowadzonego polecenia. Shell jest interpreterem poleceń co oznacza, że każde wprowadzone polecenie jest natychmiast analizowane, przekształcane na kod zrozumiały dla jądra i wykonywane. Shell pracuje więc w cyklu pobierz - analizuj - wykonaj, który można opisać następująco:

1. wysłanie znaku gotowości na terminal i oczekiwanie na polecenie użytkownika,

2. pobranie polecenia od użytkownika,

3. zdekodowanie polecenia i odszukanie w katalogach odpowiadającego mu programu,

4. przekazanie polecenia do jądra w celu utworzenia odpowiedniego procesu realizującego i oczekiwanie na jego zakończenie,

5. przyjęcie odpowiedzi od jądra i przekazanie wyniku użytkownikowi,

6. powrót do początku cyklu (pkt 1).

W systemie Unix można wprowadzić kilka komend w jednej linii oddzielonych średnikiem. Shell wykonuje je jedna po drugiej, jak gdyby wpisane były kolejno w nowych liniach.

Slajdy - wso_2_32

17. Na czym polega wykonywanie polecenia w tle (w drugim planie)?

18. Co to jest planowanie (szeregowanie) procesów?

Slajdy – wso_2_33,34

19. Podać przykłady algorytmów szeregowania procesów i wyjaśnić ich działanie?

Slajdy – wso_2_33

20. Jak rozwiązuje się zagadnienie planowania procesów w systemach typu UNIX?
21. Kiedy, po co i jak wykonywany jest proces ładowania systemu?

Proces włączania systemu komp i doprowadzenia go do takiego stanu aby mogli korzystać z niego wszyscy urzytkownicy , okreslany jest często ang terminem bootstrapping. Komputer aby mogł pracować potrzebuje systemu operacyjnego , ale startujac musu sam w jakis sposób uruchomic system operacyjny , nie mając do dyspozycji zadnego systyemu operacyjnego. Proces ładowania systemu jest w ogólnym zarysie podobny we wszystkich Unix-ach . Rozpoczyna się zawsze od wykonania instrukcji przechowywanych w pamięci ROM. Instrukcje te odpowiadają za uruchomienie programu określanego jako program boot . Program boot odpowiada za załadowanie do pamięci jądra UNIXa.
Zanim program zawarty w ROM przekaże sterowanie do boota stwierdza obecność niezbędnych urządzeń czyli dysku systemowego lub sieci. Kolejne sprawdzenie dodatkowego sprzętu , takiego jak pamięć czy główne urządzenia zewn wykonuje boot. Kiedy kontrolę nad procesem startu sys przejmuje jądro, dokonuje ono inicjalizacji swoich wewn tablic, i kontynuuje testowanie dostępnego sprzętu.
Kolejny etap – montowanie systemu plików root, sprawdzenie spójności. Wreszcie proces o numerze PID = 1 rozpoczyna wykonywanie programu init. Wszystkie pozostała pozostałe czynności przygotowujące system do normalnej pracy wykonuje proces init.

Slajdy – wso_2_35

22. Jaki proces ma PID=1, jakie zadania wykonuje?

Przodkiem wszystkich procesów w systemie jest proces o nazwie init oraz identyfikatorze PID równym 1 ,tworzony w czasie startu systemu.

Funkcja `init()`

Jest to pierwsza funkcja, która staje się procesem z prawdziwego zdarzenia. Najważniejszą sprawą jest wywołanie procedury `do_basic_setup()`, która wykonuje następujące czynności:

- zawiadamia, że wątek `init()` będzie adoptował osierocone procesy:
  `child_reaper = current;`
- rozpoczyna obsługę magistrali systemowych (ISA, PCI itp...) oraz dołączonych do nich urządzeń.
- inicjalizuje obsługę sieci `sock_init();`
- uruchamia demony:
- `/* Launch bdflush from here, instead of the old syscall way. */`
- `kernel_thread(bdflush, NULL, CLONE_FS | CLONE_FILES | CLONE_SIGHAND);`
- `kernel_thread(kupdate, NULL, CLONE_FS | CLONE_FILES | CLONE_SIGHAND);`
- `/* Start the background pageout daemon. */`
- `kswapd_setup();`
- `kernel_thread(kpiod, NULL, CLONE_FS | CLONE_FILES | CLONE_SIGHAND);`

```
kernel_thread(kswapd, NULL, CLONE_FS | CLONE_FILES | CLONE_SIGHAND);
```

- wykonuje inne niezbędne czynności:

```
/* Inicjalizacja urządzeń.. */
device_setup();

/* .. formatów plików wykonywalnych .. */
binfmt_setup();

/* .. systemów plików .. */
filesystem_setup();

/* montowanie korzenia systemu plików.. */
mount_root();
```

Po powrocie do funkcji `init()`:

- otwiera się konsolę i przekierowuje sę na nią `stdout` i `stderr`:

```
if (open("/dev/console", O_RDWR, 0) < 0)
        printk("Warning: unable to open an initial console.\n");

(void) dup(0);
(void) dup(0);
```

- wybiera się program, który zajmie się dalszym działaniem systemu. Jako pierwszy jest brany pod uwagę program wskazany w linii poleceń.

```
if (execute_command)
        execve(execute_command,argv_init,envp_init);
execve("/sbin/init",argv_init,envp_init);
execve("/etc/init",argv_init,envp_init);
execve("/bin/init",argv_init,envp_init);
execve("/bin/sh",argv_init,envp_init);
panic("No init found.  Try passing init= option to kernel.");
```

Na tym niezwykle pasjonujący ;-) proces uruchamiania systemu się kończy i rozpoczyna się nudna, codzienna praca.

No tak , żarty żartami , ale prawdziwa odpowiedz powinna iść jak sądzę jakoś tak:

Proces init odpowiada za powołanie wielu innych procesów (zawsze na drodze fork-and-exec) wśród których wiele wykonuje program getty. Kiedy procesy te mają się ku końcowi , proces rodzic zostaje o tym poinformowany przez odpowiedni sygnał . Tak więc w przypadku np. gdy urzytkownik kończy sesję , login shell wysyła do procesu init sygnał informujący go o tym ,ze właśnie kończy działanie. Init podejmuje wówczas działanie umożliwiające zalogowanie się na zwolnionym terminalu innemu użytkownikowi systemu, polegające na rozwidleniu się i wykonaniu programu getty.

23. Co to są pliki i jakie typy plików występują w systemie UNIX?

Pliki zwykłe zawierają programy, dane, listy itp. Programy systemowe, sam system, powłoka są plikami zwykłymi. Odczyt informacji z pliku nie kasuje jej. Zapisanie nowej informacji niszczy poprzednią.
Pliki specjalne, katalogi

24. Co to jest i-węzeł?

Patrz SO.pdf

I-węzeł – przechowywana na dysku struktura , opisujaca atrybuty pliku , włączając w to fizyczną lokalizacje bloków danych. W momencie przygotowywania danej partycji dyskowej na potrzeby plików ,od razu tworzona jest pewna określona liczba i-węzłow. Stanowi ona limit wyznaczający max liczbe wszelkiego rodzaju plików ,w tym plików zwykłych, katalogów , plików specjalnych oraz łączników , które będą mogły być utworzone na tym fragmencie dysku. Wszystkie i-węzły mają unikatowe numery. Z każdym plikiem związany jest opisujący go węzeł , rezerwowany w momencie tworzenia danego pliku.
W i-węźle przechowywane są wszystkie informacje o pliku z wyjątkiem nazwy pliku.

25. Jakie informacje są przechowywane w i-węźle?

Patrz SO.pdf

Są w nim przechowywane wszystkie niezbędne dane o pliku. Najważniejsze z nich to:

- Typ pliku i prawa dostępu do niego
- Liczba dowiązań.
- Identyfikator właściciela UID
- Identyfikator grupy pliku GID
- Rozmiar pliku
- Dane o adresach bloków, w których zapisany jest plik. Adresy części bloków są podawane w sposób bezpośredni. Jeśli wyczerpią się możliwości takiego adresowania, to przechodzi się na adresowanie pośrednie (tzn. podany jest adres bloku gdzie znajdują się adresy bloków składających się na plik), a jeśli to się wyczerpie, to przechodzimy na adresowanie podwójnie pośrednie ...
- Data ostatniego dostępu do pliku.
- Data ostatniej modyfikacji pliku.
- Data ostatniej modyfikacji i-węzła.

26. Jakie informacje przechowywane są w pliku typu "katalog"?

Katalogi to spisy plików zwykłych i katalogów. W linux'ie, w katalogach są zapisane jedynie dwie informacje o plikach: nazwa pliku i jego numer (i-węzeł po angielsku i-node) w systemie plików. Do pozostałych informacji (np. prawa dostępu, data ostatniej modyfikacji, adres dyskowy pliku itd.) dostęp jest umożliwiany przez i-węzeł pliku. Tak więc katalog to plik "prawie zwykły". Jedynie struktura informacji w katalogu jest z góry określona. Każdy nowy plik zwykły i każdy nowy katalog pojawiający się w systemie jest umieszczany w odpowiednim katalogu (spisie) . Usunięcie pliku zwykłego, czy też katalogu z dysku jest związane z usunięciem informacji o nich z odpowiedniego katalogu.

Katalog jest plikiem binarnym zawierającym listę plików (w tym innych katalogów) które się w nim znajdują. Każda pozycja w katalogu stanowi parę informacji złożoną z nazwy pliku oraz jego numeru i-węzła. Jest to mechanizm spajający plik opisywany przez dany i-węzeł z jego położeniem w drzewie katalogowym. Sam plik nie zawiera info o tym w jakim miejscu drzewa katalogowego jest zlokalizowany.

27. Co to jest katalog z punktu widzenia użytkownika, a co z punktu widzenia budowy systemu plików?
28. Jaka jest różnica między adresowaniem bezpośrednim a adresowaniem pośrednim?
29. Wyjaśnić mechanizm rozmieszczania bloków i fragmentów pliku w blokach na dysku?

30. Jak adresowane są bloki i fragmenty pliku na dysku?
31. Czym różni się przydział ciągły miejsca na dysku od przydziału listowego?
32. Co jest tablica FAT?
33. Porównać przydział listowy miejsca na dysku z przydziałem indeksowym?
34. Co to jest i jak jest wykorzystywana pamięć operacyjna?
35. Wyjaśnić mechanizm wymiany (swapping) procesów?

Stronicowanie i wymienianie (swapowanie) są mechanizmami za pomocą których UNIX rozdziela dostępną pamięć między aktualnie aktywne procesy, gdy ich łączne wymagania pamięciowe przekraczają fizyczną pamięć operacyjną. Swapowanie polega na zapisywaniu na dysku całego procesu ,a tym samym zwalnianiu całej związanej z nim pamięci. Proces przeniesiony do obszaru swap dysku musi zostac ponownie wczytany do pamięci , by można było go kontynuować.

36. Wyjaśnić mechanizm stronicowania na żądanie?

37. Wyjaśnić mechanizm segmentacji?

segmentacja:
adres logiczny (wirtualny) -> adres liniowy

Segmentacja.

> Adresy segmentowe są wykorzystywane do otrzymywania adresu liniowego z adresu logicznego (wirtualnego). Adres liniowy jest później przekształcany przez stronicowanie do adresu fizycznego. Każdy segment w systemie opisany jest przez ośmiobajtowy deskryptor segmentu, który zawiera niezbędne informacje (adres liniowy, wielkość, typ, prawa).

Segmenty dzielimy na dwa rodzaje:

- **Regularne** (regular segments):
  - segmenty kodu/danych
- **Systemowe**, w których wyróżniamy:
  - Task State Segments (TSS)
  ...czyli segmenty stanu procesu
  - Local/Global Descriptor Tables (LDT/GDT)
  ...czyli lokalne tablice deskryptorów

38. Podać przykłady algorytmów wymiany stron?
39. Kiedy występuje błąd strony i jak jest obsługiwany?
40. Na czym polega zarządzanie pamięcią wykonywane przez system operacyjny?

41. Jakie są zadania podsystemu wejście - wyjście?
Podsystem IO to zespół mechanizmów służących do komunikacji systemu ze światem zewnętrznym.

42. Co to są pliki specjalne i do czego służą?

Pliki specjalne w Unixie umożliwiają realizacje operacji we/wy do urządzeń.

43. Co to jest tablica rozdzielcza urządzeń we-wy, jakie informacje zawiera i do czego służy ?

44. Jak wykorzystywana jest pamięć podręczna do wydajniejszego korzystania z systemów dyskowych?

Patrz SO.pdf

45. Jakie są typy plików specjalnych?

Pliki specjale to pliki związane z urządzeniami peryferyjnymi. Zawierają stałą, nie usuwaną przez pisanie do nich, informację. Dzięki niej wiadomo co zrobić z dopisywanym do pliku specjalnego plikiem. W Linux'ie każde urządzenie peryferyjne jest również plikiem.

- Podział urządzeń na znakowe i blokowe - to jest właśnie typ urządzenia, trzeci parametr przy identyfikacji.
- Zasadnicze różnice między nimi:
    - do znakowych mamy dostęp sekwencyjny, konieczność zastosowania bufora; intuicja - nieskończony strumień; przykład - karty dźwiękowe;
    - do blokowych mamy dostęp swobodny, można jednak używać mechanizmu *cache*, aby zwiększyć wydajność; intuicja - skończony obszar podzielony na bloki; przykład - systemy plików;
- Dlaczego pliki - standaryzacja obsługi urządzenia, stosunkowo duża przenośność pomiędzy Uniksami, system plików jako pośrednik, zajmujący się ochroną na najwyższym poziomie.
- `/dev` - standardowy katalog dla plików urządzeń.
- Informacje, które możemy uzyskać nt. pliku specjalnego przy pomocy polecenia `ls` - numer nadrzędny i podrzędny, rodzaj (c - znakowe, b - blokowe); informacje te pochodzą z i-węzła.
- Operacje na plikach specjalnych - analogiczne do tych na zwykłych plikach, jednak ich obsługa jest bardziej skomplikowana.

Wyróżnimy 2 typy plików specjalnych : pliki specjalne znakowe , odpowiadające za realizacje znakowego , czyli surowego dostępu do urządzeń oraz pliki specjalne blokowe , umożliwiające dostęp blokowy. Pliki specjalne znakowe używane są w celu wykonywania niebuforowanego przesyłania danych na i z urządzenia, podczas gdy pliki sp blokowe wykorzystywane są wtedy , gdy dane mają być zapisane bądź odczytywane porcjami , określonymi jako bloki.

46. Jakie informacje są zapisane w plikach specjalnych i do czego służą?
47. Jaką rolę pełnią podprogramy obsługi urządzeń?
48. Co to jest pamięć współdzielona?

Pamięć dzielona
Na zakończenie kilka słów o pamięci dzielonej - jest to najszybszy sposób komunikacji między procesami. Polega to na tym, że ten sam obszar pamięci jest przydzielany dla kilku procesów, tak więc natychmiast po wpisaniu danych do pamięci przez jeden proces, inne procesy mogą z nich korzystać. Problem synchronizacji dostępu może być rozwiązywany za pomocą semaforów i komunikatów.

- Semafory
  W wielu sytuacjach konieczne jest uniemożliwienie dostępu do zasobów dwóm lub większej liczbie procesów. Semafor jest taką flagą uniemożliwiającą dostęp do zasobów w takich przypadkach. Jako semaforów można używć również pliki, jednak w wielu zagadnieniach byłoby to kłopotliwe.
- Komunikaty
  Następną możliwość porozumiewania się procesów to komunikaty. Jest to niewielka liczba danych,

którą można przesłać do kolejki komunikatów. Procesy mające uprawnienia mogą pobierać z tej koleki komunikaty na kilka sposobów.

Przedmiot WSO 1 1
1. Co to jest system operacyjny? 1
2. Co oznacza wielodostępność, a co wielozadaniowość? 1
3. Jakie funkcje spełnia wielodostępny system operacyjny i jakie podsystemy wchodzą w jego skład? 1
5. Co to jest proces? 1
Program a proces 2
6. Co to jest blok kontrolny procesu i do czego służy? 3
7. Co oznacza współbieżne wykonywanie procesów? 3
8. Jak powstaje nowy proces? 3
9. Jak proces jest kończony? 4
10. Jak działa funkcja systemowa exec? 4
11. Jak działa funkcja systemowa fork? 4
fork 5
13. W jakich stanach może być proces? 5
Chyba nie oto chodziło , co? Szczególnie że dotyczy to raczej Linuxa niż Unixa. 6
14. Co to jest "shell"? 7
16. W jaki sposób program shell interpretuje polecenie? 7
21. Kiedy, po co i jak wykonywany jest proces ładowania systemu? 8
22. Jaki proces ma PID=1, jakie zadania wykonuje? 8
Funkcja `init()` 8
23. Co to są pliki i jakie typy plików występują w systemie UNIX? 9
24. Co to jest i-węzeł? 9
25. Jakie informacje są przechowywane w i-węźle? 10
26. Jakie informacje przechowywane są w pliku typu "katalog"? 10
35. Wyjaśnić mechanizm wymiany (swapping) procesów? 11
37. Wyjaśnić mechanizm segmentacji? 11
Segmentacja. 11
45. Jakie są typy plików specjalnych? 12
48. Co to jest pamięć współdzielona? 12