Systemy operacyjne

Notatki zostały przygotowane na podstawie książki Marca Rochkinda „Advanced UNIX Programming (druga edycja)”.

Wprowadzenie

Rodzaje plików UNIX:

- Plik regularne
- Katalogi
- Dowiązania symboliczne
- Pliki specjalne
- Nazwany potok (pipe, FIFO)
- Gniazda

PLIKI REGULARNE
Zawierają bajty danych zorganizowane w macierz liniową i są przechowywane na dysku. Odczyt / zapis rozpoczyna się w lokalizacji bajtowej określonej przez offset pliku, który może być dowolną wartością (nawet poza końcem pliku).
Nie można wstawić / usunąć bajtów do / ze środka pliku. Dwa lub więcej procesów może jednocześnie czytać i pisać do pliku ale w takiej sytuacji wynik jest nieprzewidywalny. Aby utrzymać porządek, istnieją funkcje zajmowania plików i semafory. Pliki nie mają nazw ale i-węzły (i-node brzmi lepiej). Jest to indeks do tablicy i-nodów, przechowywanych na początku każdego obszaru dyskowego zawierającego UNIXowy system plików. I-Node zawiera następujące informacje:

- typ pliku,
- właściciel i identyfikator grupy,
- uprawnienia,
- rozmiar w bajtach,
- czas ostatniego dostępu, ostatniej modyfikacji i ostatniej zmiany statusu,
- wskaźnik do zawartości pliku.

KATALOGI I LINKI SYMBOLICZNE
Katalogi pozwalają odwoływać się do pliku po nazwie, więc prawie zawsze są używane do dostępu do plików. Każdy katalog składa się z tablicy z nazwą pliku i jego i-nodem. Ta para nazywa się linkiem. Gdy jądro systemu UNIX ma uzyskać dostęp do pliku według nazwy, automatycznie szuka w katalogu i-nodów. Jądro śledzi i-node dla bieżącej lokalizacji dla każdego z aktywnych procesów (nazywa się to ścieżką względną). Ścieżka bezwzględna

zaczyna się od „/”.

Możliwe jest, że dwa lub więcej linków odnosi się do tego samego i-noda (oznacza to, że plik może mieć więcej niż jedną nazwę). Ale nie ma w tym niejednoznaczności bo przy szukaniu po nazwach i tak znajdziemy tylko jeden i-node. Ale żeby nie było problemów z usuwaniem to każdy i-node ma licznik, który zlicza liczbę dowiązań do niego w związku z czym przy usuwaniu linku dekrementowany jest jedynie licznik i dopiero jak osiągnie on wartość 0 to wywalany jest też plik właściwy. Nie ma w sumie powodów żeby nie tworzyć wielu linków do katalogów ale ostatecznie utrudniałoby to poleceniom systemowym odnajdywanie plików więc większość kerneli na to nie zezwala.

PLIKI SPECJALNE
Najczęściej są to jakieś urządzenia, np. CD-ROM. Są dwa główne typy:

- **blokowy**
- **znakowy**

Typ blokowy zawiera tablicę bloków o skończonym rozmiarze i pulę buforów jądra żeby przyśpieszyć operacje I/O.
Typ znakowy nie posiada żadnych reguł. Mogą wykonywac opearcje zarówno na małych jak i dużych danych więc so zbyt nieregularne żeby używać bufora.
Większość systemów UNIX ma specjalny plik znakowy, który może bezpośrednio przenosić dane między przestrzenią adresową procesu a dyskiem przy użyciu bezpośredniego dostępu do pamięci (DMA), co może skutkować lepszą wydajnością rzędu wielkości.
Plik specjalny ma swojego i-node'a ale ten numer na nic nie wskazuje. Zamiast tego i-node przechowuje numer urządzenia. Jest to indeks do tabeli, której używa jądro żeby identyfikować sterowniki urzadzeń.

PROGRAMY, PROCESY I WĄTKI
Program jest ciągiem instrukcji i danych przechowywany w pliku regularnym na dysku. W swoim i-nodzie plik jest oznaczany jako plik wykonywalny, a zawartość pliku jest ułożona zgodnie z regułami ustanowionymi przez jądro. Aby uruchomić program, kernel proszony jest o utworzenie nowego procesu, który jest tak jakby środowiskiem dla programu, w którym ten jest wykonywany.

Proces składa się z trzech części: instukcji (*instruction segment*), danych użytkownika (*user data segment*) i danych systemowych (*system data segment*). Do inicjalizacji instrukcji i danych użytkownika wykorzystywany jest program. Kiedy proces jest uruchomiony to jądro śledzi jego ***wątki***, z których każdy jest osobnym strumieniem. Wszystkie potencjalne wątki czytają i zapisują te same dane procesu ale każdy wątek ma swój stos.

Dane systemowe to takie atrybuty jak aktualny katalog, deskryptory otworzonych plików, czas procesora itd. Proces nie ma dostępu i nie może modyfikować ich bezpośrednio ale jest kilka

poleceń systemowych żeby to zrobić.

Jeśli jakiś proces tworzy inny proces, to staje się on rodzicem dla procesu potomnego. Proces potomny dziedziczy większość danych systemowych rodzica, np. otwarte pliki. W przypadku wątków jest trochę inaczej, nie ma mowy o dziedziczeniu, każdy wątek ma takie same prawa dostępu do danych i zasobów

SYGNAŁY
Jądro może wysyłać sygnały do procesu. Sygnał może być inicjowany przez samo jądro, wysyłany z procesu do samego siebie, wysyłany z innego procesu lub wysyłany w imieniu użytkownika. Np. kernel może wysłać sygnał naruszenia przestrzeni adresowej, która nie jest przydzielona do procesu, który próbuje ją odczytać/zmodyfikować. Proces sam do siebie może wysłać sygnał anulowania wykonywania funkcji. Proces główny może wysłać sygnał do wszystkich procesów potomnych żeby je np. ubić. A użytkownik może np. użyć Ctrl+C i przerwać proces.

ID PROCESÓW, GROUPY PROCESÓW, SESJE
Jeśli proces nadrzędny jakiegoś procesu zostanie ubity to żeby nie stracić danych osierocony proces jest przypisywany procesowi o id = 1 (jest to proces init).
Grupa procesów to zbiór powiązanych ze sobą procesów.
Grupy procesów są później organizowane w sesje.
System UNIX generalnie tworzy jedną sesję per login. W tym kontekście grupa procesów jest nazywana ***zadaniem (job)***.
Do każdego procesu przypisane są cztery identyfikatory procesów:

- **process-ID**: dodatnia liczba całkowita, która jednoznacznie identyfikuje ten proces.
- **parent-process-ID**: ID procesu dla rodzica tego procesu.
- **ID grupy procesu**: ID procesu lidera grupy procesów. Jeśli równa się process-ID, ten proces jest liderem grupy.
- **session-leader-ID**: ID procesu lidera sesji.

KOMUNIKACJA MIĘDZYPROCESOWA
Jest bardzo wiele opcji realizowania komunikacji międzyprocesowej.

- **Dzielone wskaźniki do pliku** - rzadko używane.
- **Syngały** - nie mogą przesyłać za dużo danych na raz no i w momenecie kiedy sygnał zostanie przesłany do innego procesu to przerywa on swoje wykonywanie (co nie zawsze jest ok i komplikuje obsługę).
- **Śledzenie sygnałów** - zbyt skomplikowany i niebezpieczny żeby go używać…
- **Pliki** - w sumie spoko ale jeśli dwa procesy mają dostęp do jednego pliku to np. ten, który odczytuje dane z pliku może wyprzedzić ten, który je tam wprowadza i zakończy transmisję bo napotkał znak końca. Można niby temu jakoś zapobiec ale po co jak po paru dniach wymiany danych pliki są tak duże, że nie ma to większego sensu?

- **Pipe** - po polsku to chyba będzie łącze komunikacyjne. Jest to coś na kształ zwykłego pliku ale z tą różnicą, że istnieje mechnizm synchronizacji między odczytem a zapisem. Jeśli proces czytający dane wyjdzie za daleko przed proces je zapisujący to następuje zatrzymanie procesu czytania do momentu, aż proces zapisu nadąży. W drugą stronę jest to samo. Ma to jednak swoje wady. Po pierwsze procesy muszą być ze sobą powiązane (rodzic-dziecko lub rodzeństwo). Po drugie nie ma gwarancji, że dane są atomowe, po przepełnieniu bufora i przy wielu procesach zapisujących dane może dojść do nieoczekiwanych problemów. Po trzecie - pipy mogą być dość wolne ponieważ dane muszą zostać skopiowane z jednego procesu do kernela i stamtąd do kolejnego procesu.
- **FIFOs** - eliminuja pierwszą niedogonność pipów ale pozostałe nadal występują.
- **Semafory** - licznik, który zapobiega jednoczesnemu dostępowi do tego samego pliku przez dwa lub więcej procesów.
- **Zablokowanie pliku (*file lock*)** - podobnie jak semafory ale zabezpieczenie dostępu do konkretnego fragmentu pliku a nie do całości.
- **Komunikat** - mała ilość danych wysyłana do kolejki komunikatów. Każdy proces, który ma uprawnienia do odczytywania danych z tej kolejki może otrzymywać komuniakty.
- **Pamięć współdzielona** - potencjalnie najszybszy sposób wymiany danych między procesami. Ta sama przestrzeń pamięci jest adresowana dla dwóch procesów.
  -- **Sockety** - najbardziej zajebiste, nie muszą działać w obrębie tej samej maszyny.

Wywołania systemowe
Zazwyczaj są to małe funkcje, które wywołują specjalny kod, przenoszą kontrolę z procesu użytkownika do kernela i później zwracają wynik. Przez to, że wywoływane są dwa przełączenia pomiędzy kontekstami (z procesu do kernela i z powrotem) to zajmuje to więcej czasu niż wywołanie funkcji z przestrzeni adresowej procesu.

Obsługa błędów

Procesy i wątki

Kiedy program UNIXowy jest wywoływany to dostaje dwie kolekcje danych od procesu, który go wywołał: argumenty i środowisko. Do programów napisanych w C przekazywane są one w postaci tablic char* oprócz ostatniego NULa, który kończy tablice argv. Do programu przekzywana jest również liczba parametrów. Przykład z C:

```
int main(
    int argc,       /* argument count */
    char *argv[]    /* array of argument strings */
)
```

argc nie uwzględnia ostatniego NULa.

Pobranie wartości zmiennej środowiskowej:

```
#include <stdlib.h>

inf main(void) {
    char *s = getenv("LOGNAME");
    return 0;
}
```

Żeby ustawić zmienną środowiskową używa się polecenia **setenv**, żeby ją usunąć **unsetenv**.

Wywołanie **exec** ponownie inicjalizuje proces z wyznaczonego programu. Program się zmienia w trakcie wywoływania procesu. Natomiast **fork** tworzy nowy proces z dokładną kopią istniejącego kopiując instrukcje, dane użytkownika i dane systemowe. Nowy proces nie jest inicjalizowany więc zarówno stary jak i nowy wykonują te same instrukcje. W ogólnym rozrachunku stosowanie ich osobno jest bardzo ograniczone, najczęściej wykorzystywane są razem.

Właściwie nie ma wywołania systemowego o nazwie „exec”. Tak zwane wywołania systemowe „exec” są zestawem sześciu, z nazwami postaci **execAB**, gdzie A jest l lub v, w zależności od tego, czy argumenty są bezpośrednio w wywołaniu (liście) czy w tablicy (wektor), a B jest albo nieobecny, albo wynosi p i wskazuje, że zmienna środowiskowa PATH powinna być używana do wyszukiwania programu, albo e, aby wskazać, że dane środowisko ma być używane. Tak więc tymi sześcioma wywołaniami są:

- **execl**
- **execv**
- **execlp**
- **execvp**
- **execle**
- **execve**