1 Wstęp

1.1 Czym jest system operacyjny ?

System operacyjny jest programem, który działa jako pośrednik pomiędzy użytkownikiem a sprzętem. Zadaniem systemu jest stworzenie środowiska do uruchamiania programów. Oprogramowanie działające bez SO to **firmware** (BIOSy, oprogramowanie do telefonów komórkowych itp.)

1.2 Ewolucja systemów operacyjnych

1.2.1 Systemy wsadowe

Program, dane i sterowanie wczytywane były z kart perforowanych, gdzie jedna karta odpowiadała jednej linijce programu. Następowała kompilacja programu, a następnie wydruk wyników lub błędów programu. Zadaniem systemu było zrozumieć program na kartach, dokonać obliczeń, wydrukoawć wyniki lub błędy i przejść do następnego zadania Zadania wykonywane były kolejno - następne zadanie po zakończeniu poprzedniego, a ich kolejność ustawiał operator. Zadania o podobnych wymaganiach grupowane były w tzw. *batche*, czyli wsady.

Zalety:

- Bardzo prosty system operacyjny (tylko automatyczne przekazywanie sterowania od jednego zadania do drugiego)

Wady:

- Brak nadzoru użytkownika podczas wykonywania zadania
- Tylko jedno zadanie w tym samym czasie
- Małe wykorzystanie CPU - bezczynność podczas wczytywania i wydruku danych

1.2.2 Spooling, czyli równoczesne wykonywanie operacji I/O i obliczeń

SPOOL to skrót od angielskiego *Simultaneus Peripheral Operations On Line*, czyli równoczesnego wykonywania operacji I/O oraz obliczeń. Spooling umożliwia wykonanie w tym samym czasie wczytywania lub wydruku danych jednego zadania oraz obliczeń do innych zadań. Kosztem zajęcia części dysku stało się możliwe lepsze wykorzystanie zasobów CPU oraz urządzeń peryferyjnych.

1.2.3 Wieloprogramowy system wsadowy

Wieloprogramowy system wsadowy był rozwinięciem systemu wsadowego. Umożliwiał on równoczesne wykonywanie kilku zadań - gdy zadanie oczekiwało na operację I/O lub zostało zajęte operacją I/O, system przełączał się na liczenie innego zadania. System powracał do poprzedniego zadania, gdy aktualnie liczone przechodziło w tryb oczekiwania na I/O, a pierwsze zakończyło obsługę I/O

Zalety:

- Lepsze wykorzystanie CPU i I/O

- Szybsze wykonywanie puli zadań

Wady:

- Większe skomplikowanie SO (planowanie zadań, przydział CPU itp.)
- Zadanie bez operacji I/O mogło na długo zablokować procesor

1.2.4 Systemy z podziałem czasu

Systemy z podziałem czasu charakteryzują się tym, że CPU wykonuje na przemian wiele różnych zadań, przełączanych na tyle szybko, że użytkownicy mogą współdziałać z każdym programem podczas wykonywania. Każdy użytkownik ma wrażenie, że ma do dyspozycji cały komputer, chociaż jest on dzielony pomiędzy wielu użytkowników. Systemy takie oprócz planowania zadań muszą posiadać mechanizmy zarządzania pamięcią i dostępem do systemu plików oraz synchronizowania zadań i komunikacji między nimi

1.2.5 Systemy równoległe

Systemy równoległe charakteryzują się tym, że wiele procesorów współpracuje ze sobą, dzieląc szynę komputera, zegar, pamięć oraz urządzenia I/O. W przypadku awarii jednego z procesorów system rozdziela zadania na pozostałe, co gwarantuje niezawodność systemu. Systemy takie dzielimy na symetryczne (SMP - *Symmetric Multi Processing*) oraz asymetryczne (AMP - *Asymmetric Multi Processing*),.

1.2.6 Systemy rozproszone *(cluster)*

Systemy rozproszone, czyli klastry (z ang. *cluster*) są to systemy rozdzielające zadania pomiędzy kilka (kilkanaście, a nawet kilkaset) komputerów spiętych siecią. Do systemów takich zaliczamy systemy obliczeniowe (np. *SETI@HOME*), serwerowe itp. Po co jednak dzielić zadania pomiędzy kilka komputerów?

- Podział zasobów
- Przyspieszenie obliczeń przez działanie współbieżne
- Niezawodność - gdy „padnie” jeden z komputerów, pozostałe liczą dalej, przejmując jego obliczenia
- Komunikacja międzyludzka - administratorzy komputerów mogą się porozumiewać ze sobą i rozplanowywać zadania

1.2.7 Systemy czasu rzeczywistego *(RTS)*

Systemy czasu rzeczywistego (z ang. RTS - *Real Time Systems*) są to systemy stosowane tam, gdzie istnieją surowe wymagania dotyczące czasu lub przepływu danych takie jak na przykład systemy nadzoru eksperymentów, czy sterowniki i programatory maszyn. Systemy takie wykonują wszystkie zadania w czasie rzeczywistym, z reguły bez podziału czasu, minimalizując czas każdej operacji. Dzielimy je na:

1. Systemy rygorystyczne - minimalizacja czasu operacji, podział czasu absolutnie wykluczony
2. Systemy łagodne - możliwe do aplikacji w standardowych warunkach, np. *QNX*

2 Działanie systemu operacyjnego

2.1 Przerwania

Przerwanie jest to sygnał pochodzący od sprzętu lub programu, sygnalizujący wystąpienie zdarzenia. Sygnały przerwań od sprzętu są wysyłane do procesora przez specjalne linie. Oprogramowanie powoduje przerwania poprzez **wywołania systemowe**.

Zdarzenia powodujące przerwania:

- Zakończenie operacji I/O
- Dzielenie przez 0
- Niedozwolony dostęp do pamięci
- Zapotrzebowanie na usługę systemu
- ...

Każdemu przerwaniu odpowiada **procedura obsługi**. Podczas obsługi przerwania należy zapamiętać adres przerwanego rozkazu, zawartości rejestrów itp. W nowych systemach adres powrotny znajduje się na stosie systemowym. Podczas obsługi przerwania inne przerwania są blokowane, chyba że mają wyższy priorytet (**przerwania maskowane**).

2.1.1 Wektor przerwań

Aby przyspieszyć operacje obsługi przerwań stosuje się tablice wskaźników do procedur obsługujących poszczególne przerwania. Indeksy tej tablicy odpowiadają numerom przerwań, a elementami tablicy są adresy procedur obsługi przerwania.

2.1.2 Pułapki, wyjątki

Wyjątek jest to przerwanie generowane przez oprogramowanie spowodowane np. przez błąd numeryczny, niewłaściwy dostęp do pamięci lub generowane na życzenie użytkownika.

2.2 Obsługa operacji wejścia/wyjścia

Obsługa operacji I/O w systemie może być dwojaka:

Synchroniczna - system podczas obsługi I/O przerywa operacje do czasu jej zakończenia

Asynchroniczna - podczas obsługi I/O system zajmuje się czymś innym, a nie czeka bezowocnie. Taka transmisja jest stosowana powszechnie w nowoczesnych systemach.

2.3 Tryby pracy procesora

W nowoczesnych systemach rozróżniamy dwa tryby pracy procesora:

Tryb użytkownika (*user-mode*) - jest to tryb ograniczony - nie można w nim wykonywać niebezpiecznych dla systemu operacji uprzywilejowanych

Tryb nadzorcy (*kernel-mode*) - tryb uprzywilejowany, w którym wykonywane są potencjalnie niebezpieczne dla systemu operacje. Użytkownik nie może samodzielnie przełączyć procesora w tryb nadzorcy - może to zrobić jedynie sam system.

2.4 Procesy

Proces jest to program, który jest aktualnie wykonywany. Proces jest podstawową jednostką pracy w SO, a sam system jest zbiorem procesów systemowych oraz procesów użytkownika. Wykonanie procesu musi przebiegać w sposób sekwencyjny (w dowolnej chwili na zamówienie naszego procesu może być wykonany co najwyżej jeden rozkaz programu).

2.4.1 Zarządzanie procesami przez SO

System ma za zadanie:

- Tworzyć i usuwać procesy
- Wstrzymywać i wznawiać procesy
- Dostarczać mechanizmów synchronizacji procesów (o tym później)
- Dostarczać mechanizmów komunikacji procesom (IPC - *Inter-Process Communication* - o tym później)
- Dostarczać mechanizmów obsługi zakleszczeń

2.4.2 Składniki procesu

Proces składa się z następujących części:

- kod programu (sekcja tekstu),
- bieżąca czynność (wskazana przez licznik rozkazów),
- zawartość rejestrów procesora,
- stos procesu,
- sekcja danych.

2.4.3 Stany procesu

Proces może mieć następujący stan:

- nowy - proces został utworzony.
- aktywny - są wykonywane instrukcje,
- oczekujący - czeka na wystąpienie zdarzenia, np. zakończenie operacji wejścia-wyjścia,
- gotowy - czeka na przydział procesora,
- zakończony - zakończył działanie.

2.4.4 Blok kontrolny procesu

Blok kontrolny procesu jest to część odpowiedzialna za przechowywanie informacji o procesie. Składa się on zwykle z następujących części:

- Numer procesu
- Aktualny stan procesu
- Licznik rozkazów - adres następnego rozkazu do wykonania
- Rejestry
- Adresy pamięci
- Wykaz otwartych plików
- Zarządzanie pamięcią (granice pamięci, tablice stron, tablice segmentów...)
- Informacje do rozliczeń (zużyty czas procesora, czas całkowity, konta...)
- Informacje o stanie wejścia-wyjścia (urządzenia przydzielone do procesu, wykaz otwartych plików itd.)
- Informacje do planowania przydziału procesora (priorytet procesu, wskaźniki do kolejek)
- ...

2.4.5 Kolejki procesów

Kolejki procesów (*process queues*) mogą być tworzone przez system w celu grupowania np. procesów oczekujących na wykonanie. Kolejki takie dzielimy na:

- Kolejki zadań (job queue) - tworzą ją procesy wchodzące do systemu
- Kolejki procesów gotowych (ready queue) - procesy gotowe do działania, umieszczone w pamięci
- Kolejki do urządzeń (device queue) - procesy czekające na konkretne urządzenie

2.4.6 Planiści

Planiści służą do planowania przydzielania poszczególnym procesom zasobów systemowych. Planistów dzielimy na dwie kategorie:

Planista długoterminowy (planista zadań) - wybiera procesy do kolejki procesów gotowych, do pamięci. Jest on wywoływany stosunkowo rzadko (sekundy) i nie musi być szybki. Zadaniem planisty długoterminowego jest dobór optymalnej mieszanki zadań ograniczonych przez procesor (wymagających więcej czasu procesora niż I/O) i przez we-wy (wymagających obsługi I/O częściej niż procesora)

Planista krótkoterminowy (planista przydziału procesora) - wybiera proces z puli procesów gotowych i przydziela mu procesor. Jest on wywoływany bardzo często (co ms) i musi być bardzo szybki.

Planista średnioterminowy - występuje w niektórych systemach z podziałem czasu. Jego zadaniem jest, w koniecznych przypadkach, zmniejszanie stopnia wieloprogramowości poprzez wysyłanie części zadań chwilowo na dysk (*swapping*). Pomaga to w doborze lepszego zestawu procesów w danej chwili, lub dla zwolnienia obszaru pamięci.

2.4.7 Przełączanie kontekstu

Podczas przejścia procesora z wykonywania jednego procesu do drugiego należy przechować stan starego procesu i załadować przechowany stan nowego. Z punktu widzenia systemu są to działania nieproduktywne, tak jak przygotowanie czy sprzątanie stanowiska pracy, ale są niezbędne przy wieloprogramowości. Mechanizm wątków pozwala na redukcję czasu przełączania kontekstu.

2.4.8 Tworzenie procesu

Proces macierzysty tworzy potomne za pomocą funkcji systemowej. Nowy proces też może tworzyć potomne - powstaje wtedy drzewo procesów (ładnie widoczne w systemach UNIXowych po wywołaniu komendy *pstree*). Proces macierzysty i potomek mogą dzielić w całości, w części, lub wcale nie dzielić ze sobą zasobów. Proces macierzysty i potomek działają równolegle, lub też proces macierzysty czeka, aż potomek zakończy działanie.

Proces potomny może być kopią procesu macierzystego, lub otrzymać zupełnie nowy program.

W systemie UNIX:

Nowy proces tworzy się funkcją systemową *fork()*. Potomek zawiera kopię przestrzeni adresowej przodka - daje to możliwość komunikacji pomiędzy procesami.

Funkcja systemowa *execve()* ładuje nowy program do przestrzeni adresowej procesu (niszcząc poprzednią zawartość) i rozpoczyna jego wykonywanie.

Proces macierzysty albo tworzy nowych potomków, albo czeka na zakończenie procesu potomnego.

W systemie VMS: Tworzy się nowy proces, umieszcza w nim nowy program rozpoczyna jego wykonywanie.

W systemie Windows NT: Występują obydwa mechanizmy - albo tworzona jest kopia przestrzeni adresowej przodka, albo ładowany jest nowy program.

2.4.9 Zakończenie procesu

Po wykonaniu ostatniej instrukcji proces prosi system operacyjny o usunięcie przy pomocy funkcji systemowej *exit()*. System przekazuje wyniki działania potomka do procesu macierzystego (wykonującego funkcję systemową *wait()*), a następnie odbiera potomkowi wszystkie zasoby (pamięć, otwarte pliki, bufory).

Proces macierzysty może spowodować „awaryjne" zakończenie potomka w przypadku gdy:

- Potomek nadużył któregoś z przydzielonych zasobów,
- Wykonywane przez potomka zadanie stało się zbędne,
- Proces macierzysty kończy się, a system nie zezwala na działanie „sieroty".

2.4.10 Procesy współpracujące

Procesy są współpracujące, jeżeli „nasz" proces może wpływać na inne procesy, a inne procesy mogą wpływać na niego. Zalety takiego rozwiązania są następujące:

- Dzielenie informacji - kilka procesów może korzystać z danych np. z jednego pliku,
- Przyspieszenie obliczeń - w systemach wieloprocesorowych istnieje możliwość podziału zadania na mniejsze podzadania, wykonywane równolegle
- Modularność - można konstruować system w sposób modularny
- Wygoda - jeden użytkownik może w tym samym czasie wykonywać kilka zadań, np. edycję, kompilację, drukowanie.

2.5 Komunikacja międzyprocesowa

Komunikacja międzyprocesowa (IPC - *Inter Process Communication*) jest mechanizmem zapewniającym ciągłą wymianę informacji pomiędzy procesami współpracującymi.

W systemie komunikatów występują tylko dwie operacje - nadaj komunikat oraz odbierz komunikat. W celu realizacji komunikacji procesy muszą ustanowić łącze komunikacyjne oraz móc nadawać i odbierać komunikaty.

Istnieje kilka metod komunikacji międzyprocesowej w rożnych systemach operacyjnych, jednak z grubsza IPC można podzielić na dwie kategorie:

Komunikacja bezpośrednia - łącze jest w tym wypadku dwukierunkowe, dokładnie jedno pomiędzy dwoma procesami, dzielone tylko przez dwa procesy komunikujące się ze sobą. Wystarczy, aby procesy te znały swoje identyfikatory.

Komunikacja pośrednia - komunikaty są nadawane i odbierane za pomocą skrzynek pocztowych (portów). Procesy mogą się z sobą skomunikować jeżeli mają wspólną skrzynkę pocztową. Łącze jest ustanawiane jedynie wtedy, gdy procesy dzielą jakąś skrzynkę, może być związane z więcej niż dwoma procesami. Każda para procesów może mieć kilka łączy, poprzez różne skrzynki pocztowe, łącze może być jedno lub dwukierunkowe. Może jednak wystąpić w tym wypadku **problem trzech procesów**.

2.5.1 Problem trzech procesów

Trzy procesy P1, P2 i P3 dzielą jedną skrzynkę pocztową. Proces P1 wysyła komunikat, natomiast P2 i P3 próbują go odebrać - powstaje konflikt.

Jak tego uniknąć?

- Zezwalać jedynie na łącza pomiędzy dwoma procesami,
- Zezwalać co najwyżej jednemu procesowi na wykonanie w danej chwili operacji odbioru,
- Dopuścić, aby system wybrał proces do którego dotrze komunikat (albo P2 albo P3, a nie oba). System powinien poinformować nadawcę o wyborze.

2.5.2 Buforowanie komunikatów

System może przechowywać oczekujące komunikaty w buforze, zwanym **kolejką komunikatów**. Kolejka komunikatów może mieć następujące pojemności:

Pojemność zerowa - łącze nie dopuszcza aby czekał w nim jakikolwiek komunikat - nadawca czeka aż odbiorca odbierze,

Pojemność ograniczona - w kolejce może pozostawać tyle komunikatów, na ile zaprojektowano kolejkę. W przypadku kolejki pełnej nadawca musi czekać.

Pojemność nieograniczona - kolejka ma potencjalnie nieskończoną długość. Nadawca nigdy nie czeka.

2.5.3 Komunikacja - sytuacje awaryjne

Zakończenie procesu - system musi być w stanie rozwiązać problemy:

- gdy proces czeka na komunikaty z zakończonego,
- gdy nadaje komunikaty do zakończonego,

Utrata komunikatów

- system wykrywa to i ponownie nadaje komunikat,
- proces nadawczy wykrywa i ew. powtarza komunikat,
- system wykrywa i powiadamia proces nadawczy.

Zniekształcenie komunikatów - kontrola poprawności przez sumy kontrolne, sprawdzanie parzystości itd.

2.6 Wątki

Wątek (proces lekki) jest podstawową jednostką wykorzystania procesora. Działanie wątków przypomina działanie procesów. Mogą być w stanach: gotowości, zablokowania, aktywności, kończenia. Wątek może tworzyć wątki potomne, może się zablokować do czasu wykonania wywołania systemowego. Jeśli jeden wątek jest zablokowany, może działać inny wątek. Wątki jednego zadania są do siebie zależne - mogą np. nadpisywać stosy innych wątków. Ale z drugiej strony - producent i konsument mogą być wątkami jednego zadania, wspólny obszar danych znacznie zwiększy wydajność procesu.

Wątek składa się z:

- licznika rozkazów,
- zbioru rejestrów,
- obszaru stosu

Takie elementy jak sekcja kodu, sekcja danych, czy zasoby systemu (otwarte pliki, sygnały) są wspólne dla kilku równorzędnych wątków.

Zalety użycia wątków:

- Przełączanie między wątkami i tworzenie nowych wątków nie wymaga dużej aktywności procesora.
- Przy przełączaniu nie trzeba wykonywać prac związanych z zarządzaniem pamięcią

2.6.1 Poziomy wątków

Wątki poziomu jądra posiadają małą strukturę danych i stos, podlegają planowaniu, przełączanie ich nie wymaga informacji o pamięci i jest stosunkowo szybkie. Procesy lekkie posiadają blok kontrolny procesu, potrzebne informacje o pamięci, przełączanie kontekstu dość wolne. Wątki użytkownika posiadają stos i licznik rozkazów, są przełączanie szybko, gdyż jądro nie jest angażowane.

2.7 Zarządzanie pamięcią przez SO

Do najważniejszych zadań systemu w zakresie zarządzania pamięcią należą:

- Ewidencja aktualnie zajętych obszarów pamięci, dostarczanie informacji o użytkownikach danego obszaru
- Decydowanie o tym, które procesy mogą być załadowane do zwolnionych obszarów pamięci
- Przydzielanie i zwalnianie obszarów pamięci stosownie do potrzeb

2.7.1 Bezpośredni dostęp do pamięci (*DMA*)

W przypadku wolnych urządzeń I/O obsługa przesyłania danych z bufora urządzenia do pamięci nie angażuje zbytnio procesora. Dla urządzeń szybkich (dysk, sieć) wygodniej jest przesyłać cały blok danych bezpośrednio do pamięci, bez angażowania procesora. Umożliwia to mechanizm *Direct Memory Access*, realizowany sprzętowo. **Uwaga! Kradnie cykle pamięci!**

2.8 Zarządzanie plikami przez SO

Do zadań systemu w tym zakresie należą:

- Tworzenie i usuwanie plików oraz katalogów
- Dostarczanie informacji do manipulacji plikami i katalogami
- Odwzorowywanie plików na obszary pamięci pomocniczej
- Składowanie plików na trwałych nośnikach pamięci

2.9 Inne funkcje SO

- Zarządzanie systemem I/O (spooling, buforowanie, pamięć $\cdots$)
- Zarządzanie pamięcią pomocniczą dyskową (swap, cache $\cdots$)

- Praca sieciowa
- System ochrony
- System interpretacji poleceń (powłoka)

2.10 Funkcje systemowe

Funkcje systemowe tworzą interfejs pomiędzy wykonywanym programem a SO. Poprzez funkcje systemowe użytkownik „daje polecenia" systemowi. Funkcje systemowe mają następujące zadania:

2.10.1 Nadzorowanie procesów

- Załadowanie lub wykonanie programu,
- Zakończenie lub zaniechanie procesu,
- Utworzenie lub zakończenie procesu (potomnego),
- Pobranie lub ustawienie parametrów procesu,
- Czekanie czasowe,
- Oczekiwanie na zdarzenie lub sygnalizacja zdarzenia,
- Przydział i zwolnienie pamięci.

2.10.2 Operacje na plikach

- Utworzenie lub usunięcie pliku,
- Otwarcie lub zamknięcie pliku,
- Czytanie, pisanie lub zmiana położenia,
- Pobranie lub ustawienie atrybutów pliku.

2.10.3 Operacje na urządzeniach

- Zamówienie lub zwolnienie urządzenia,
- Czytanie, pisanie lub zmiana położenia,
- Pobranie lub ustawienie atrybutów urządzenia,
- Logiczne przyłączanie lub odłączanie urządzeń.

2.10.4 Utrzymywanie informacji

- Pobranie lub ustawienie daty/czasu,
- Pobranie lub ustawienie danych systemowych,
- Pobranie atrybutów procesu, pliku lub urządzenia,
- Ustawienie atrybutów procesu, pliku lub urządzenia.

2.10.5 Komunikacja

- Utworzenie, usunięcie połączenia komunikacyjnego,
- Nadawanie, odbieranie komunikatów,
- Przekazywanie informacji o stanie,
- Przyłączanie i odłączanie urządzeń zdalnych.

2.11 Maszyna wirtualna

Jest to oprogramowanie, które systemom i programom użytkowym udostępnia wirtualny sprzęt. Przykładem takiego oprogramowania są emulatory, np. systemu IBM PC na SPARC czy Mac. Mechanizm ten wykorzystuje też wirtualna maszyna *Javy*.

3 Planowanie przydziału procesora w SO

Ogólnie rzecz biorąc, planowanie przydziału procesora w systemie polega na tym, że jeśli w pamięci operacyjnej znajduje się kilka procesów jednocześnie i jakiś proces musi czekać, system operacyjny odbiera mu procesor i oddaje do dyspozycji innego procesu.

Planowanie przydziału procesora jest podstawową funkcją każdego systemu operacyjnego.

Proces ograniczony przez wejście-wyjście ma zazwyczaj dużo krótkich faz procesora. Proces ograniczony przez procesor może mieć mało, lecz bardzo długich faz procesora.

3.1 Planowanie wywłaszczeniowe i niewywłaszczeniowe

Decyzje o zmianie przydziału procesora podejmowane są, gdy:

1. proces przeszedł od stanu aktywności do czekania, np. na zakończenie potomka, lub zamówił we-wy,
2. proces przeszedł od stanu aktywności do gotowości, np. wskutek wystąpienia przerwania,
3. proces przeszedł od stanu czekania do gotowości, np. po zakończeniu we-wy,
4. proces kończy działanie.

Jeśli planowanie odbywa się tylko w przypadkach 1 i 4, to mamy do czynienia z planowaniem **niewywłaszczeniowym**, w przeciwnym wypadku planowanie jest **wywłaszczeniowe**.

3.2 Ekspedytor (*dispatcher*)

Ekspedytor jest to moduł, który przekazuje procesor do dyspozycji procesu wybranego przez planistę krótkoterminowego. Do jego zadań należy:

- przełączanie kontekstu,
- przełączanie procesu do trybu użytkownika,
- wykonanie skoku do odpowiedniego adresu w programie w celu wznowienia działania programu

Opóźnienie ekspedycji - czas, jaki ekspedytor zużywa na wstrzymanie jednego procesu i uaktywnienie innego. Czas ten powinien być możliwie najkrótszy - ekspedytor powinien więc być jak najszybszy.

3.3 Kryteria planowania przydziału

Wykorzystanie procesora - w realnych systemach od 40% (słabe) do 90% (intensywne),

Przepustowość - mierzona ilością procesów kończonych w jednostce czasu (dla długich - kilka na godzinę, dla krótkich - kilka na sekundę),

Czas cyklu przetwarzania - od nadejścia procesu do systemu, do jego zakończenia (suma czasów oczekiwania na wejście do pamięci, w kolejce procesów gotowych, okresów aktywności i operacji wejścia-wyjścia)

Czas oczekiwania - suma czasów w których procesor czeka w kolejce p. gotowych,

Czas odpowiedzi - w systemach interakcyjnych - czas od wysłania żądania do rozpoczęcia odpowiedzi.

Planowanie jest **optymalne**, gdy następujące kryteria są spełnione:

- maksymalne wykorzystanie procesora,
- maksymalna przepustowość,
- minimalny czas cyklu przetwarzania, minimalny czas oczekiwania,
- minimalny czas odpowiedzi.

3.4 Metody planowania

3.4.1 Metoda FCFS

Metoda FCFS jest to metoda *first come - first served*. Jest to metoda, oględnie mówiąc „kto pierwszy, ten lepszy". System obsługuje procesy w kolejności przychodzenia. Planowanie metodą FCFS nie jest optymalne, gdyż średni czas oczekiwania bardzo zależy od kolejności zgłoszenia procesów. Występuje tu także *efekt konwoju* - małe procesy czekają na zwolnienie procesora przez jeden wielki proces. Algorytm FCFS jest niewywłaszczający. Jest on nieużyteczny w systemach z podziałem czasu, w których procesy powinny dostawać procesor w regularnych odstępach czasu.

3.4.2 Metoda SJF

Metoda SJF jest metodą *shortest job first*. System przydziela procesor najpierw najkrótszym zadaniom. Można udowodnić, że planowanie tą metodą jest optymalne, gdyż umieszczenie krótkiego procesu przed długim w większym stopniu zmniejsza czas oczekiwania krótkiego procesu niż zwiększa czas oczekiwania procesu długiego. Algorytm SJF jest często używany przy planowaniu długoterminowym. Problemem jest to, że nie można przewidzieć długości następnej fazy procesora, można ją tylko szacować, najczęściej za pomocą średniej wykładniczej z poprzednich faz procesora. Algorytm SJF może być wywłaszczający lub niewywłaszczający - Gdy do kolejki dochodzi nowy proces, który posiada fazę procesora krótszą niż pozostały czas w bieżącym procesie, algorytm wywłaszczający odbiera procesor bieżącemu procesowi i przekazuje go krótszemu. Metoda ta nazywa się SRTF (*shortest remaining time first*) czyli „najpierw najkrótszy pozostały czas". Algorytm niewywłaszczający pozwala na dokończenie fazy procesora.

3.4.3 Planowanie priorytetowe

Mechanizm bardzo podobny do SJF, ale kryterium szeregowania jest priorytet procesu. Najpierw wykonywane są procesy o ważniejszym priorytecie. Priorytety mogą być definiowane wewnętrznie, na podstawie pewnych cech procesu (np. wielkość pamięci, limity czasu, zapotrzebowanie na urządzenia we-wy itd..)

Priorytety definiowane zewnętrznie mogą np. zależeć od ważności użytkownika, jego firmy czy też od innych politycznych uwarunkowań.

Podstawową wadą jest to, że procesy o niskim (mało ważnym) priorytecie mogą nigdy nie dostać procesora (głodzenie, nieskończone blokowanie) Przykład: w 1973 r. w wycofywanym z eksploatacji na MIT komputerze IBM 7094 wykryto „zagłodzony" proces o niskim priorytecie, który został zapuszczony do wykonania w 1967 roku (*sic!*)

Rozwiązaniem problemu jest „postarzanie" czyli podnoszenie priorytetu procesów oczekujących zbyt długo. Przykład: przydział mieszkania dla dziadka w *Alternatywy 4*.

3.4.4 Planowanie RR

Planowanie RR jest to tzw. planowanie rotacyjne (RR - *round-robin*) Zaprojektowane zostało specjalnie do systemów z podziałem czasu. Każdy proces otrzymuje kwant czasu (10-100ms), po upływie którego jest wywłaszczany i umieszczany na końcu kolejki zadań gotowych. Kolejny proces do wykonania jest wybierany zgodnie z algorytmem FCFS Jeżeli jest n procesów gotowych a kwant czasu wynosi q, to każdy proces czeka nie dłużej niż $(n-1) \cdot q$ jednostek czasu. Wydajność algorytmu bardzo zależy od kwantu czasu q Gdy q jest duże, to algorytm RR przechodzi w FCFS Gdy q jest bardzo małe, to znaczna część czasu procesora poświęcona jest na przełączanie kontekstu

Dobrą metodą jest takie przyjęcie kwantu czasu q, by 80% faz procesora było krótsze niż jeden kwant czasu.

3.4.5 Kolejki wielopoziomowe

Kolejka procesów gotowych jest podzielona na oddzielne podkolejki, na przykład: kolejka procesów pierwszoplanowych (foreground), kolejka procesów drugoplanowych (background).

Przykładowe proponowane algorytmy planowania:

- procesy pierwszoplanowe: RR
- procesy drugoplanowe: FCFS

Procesy pierwszoplanowe mają absolutny priorytet nad drugoplanowymi. Aby nie „zagłodzić" procesów 2-planowych, stosuje się podział czasu na kolejki, przykładowo 80% czasu procesora dla kolejki pierwszoplanowej, ipozostałe 20% dla drugoplanowej.

3.4.6 Kolejki wielopoziomowe ze sprzężeniem zwrotnym

Mechanizm ten pozwala na przesuwanie procesów pomiędzy kolejkami Proces, który używa za dużo procesora, można „karnie" przenieść do kolejki o niższym priorytecie i przez to dać szerszy dostęp do procesora innym procesom. Dzięki temu procesy ograniczone przez we-wy i procesy interakcyjne mogą pozostać w kolejkach o wyższch priorytetach. Długo oczekujące procesy z kolejki niskoprioritetowej mogą być przeniesione do ważniejszej - działa mechanizm postarzania procesów (przeciwdziała ich głodzeniu). Planowanie ze sprzężeniem zwrotnym jest najbardziej złożonym algorytmem planowania przydziału procesora.

Przykład:

Kolejka trzypoziomowa: K0, K1,K2

Proces wchodzący trafia do kolejki k0 i dostaje kwant czasu 8 ms.

Jeśli nie zakończy się w tym czasie, jest wyrzucany na koniec niższej kolejki K1.

Gdy kolejka K0 się opróżni i przyjdzie czas wykonywania naszego procesu, dostaje on kwant czasu 16 ms.

Jeśli i w tym czasie proces nie skończy działania, jest wyrzucany na koniec kolejki K2, obsługiwanej w porządku FCFS (oczywiście pod warunkiem, że kolejki K0 i K1 są puste).

Tak więc najszybciej wykonywane są procesy do 8 ms, nieco wolniej procesy od 8 do $8 + 16 = 24ms$, a najdłużej czekają procesy długie (są obsługiwane w cyklach procesora nie wykorzystanych przez kolejki 1 i 2)

3.5 Planowanie wieloprocesorowe

Metody planowania przydziału CPU dla wielu procesorów możemy podzielić na dwie kategorie:

Planowanie heterogeniczne - dla systemów sieciowych, rozproszonych, o różnych procesorach - bardzo trudne w realizacji

Planowanie homogeniczne - dla procesorów tego samego typu. Nie stosuje się oddzielnych kolejek dla każdego procesora - możliwość przestojów niektórych procesorów.

Przetwarzanie danych przez wiele procesorów możemy podzielić na:

Wieloprzetwarzanie symetryczne (SMP) - każdy procesor sam wybiera procesy ze wspólnej kolejki - działanie musi być bardzo starannie zaprogramowane, aby uniknąć np. dublowania.

Wieloprzetwarzanie asymetryczne (AMP) - jeden procesor jest nadrzędny (master) i on planuje przydział procesów, a pozostałe (slave) wykonują przydzielone im zadania.

3.6 Planowanie w czasie rzeczywistym

W rygorystycznych systemach czasu rzeczywistego musi być zapewnione wykonanie zadania krytycznego w określonym czasie. Następuje rezerwacja zasobów niezbędnych do wykonania zadania. Jeśli planista nie może zarezerwować zasobów - rezygnuje z przydziału zadania do procesora.

W łagodnych systemach czasu rzeczywistego procesy cz. rz. mają wyższy priorytet niż pozostałe zadania. Priorytet tych procesów nie może ulec obniżeniu. W krytycznych przypadkach musimy zezwolić na wywłaszczanie funkcji systemowych (muszą one posiadać punkty wywłaszczenia), lub nawet całego jądra (potrzebne mechanizmy synchronizacji danych jądra). Wysokopriorytetowe procesy nie mogą czekać na zakończenie niskopriorytetowych.

3.7 Ocena algorytmów planowania

Modelowanie deterministyczne - zakładamy z góry konkretne obciążenie systemu i definiujemy zachowanie się poszczególnych algorytmów w tych warunkach. Przykład - wyznaczanie średniego czasu oczekiwania z diagramu Gantta

Modele obsługi kolejek - bazują na badaniach dot. analizy obsługi kolejek w sieciach.

Wzór Little'a: $n = L \cdot W$ gdzie: n - średnia długość kolejki, L - ilość nowych procesów na sekundę, W - średni czas oczekiwania w kolejce.

Symulacje - programuje się model systemu i generuje dane symulacyjne. Taśmy śladów z rzeczywistego systemu pomagają w ustawieniu danych do symulacji

Implementacja - algorytm zaimplementowany w rzeczywistym systemie. Metoda kosztowna ale dokładna.

4 Synchronizacja procesów

4.1 Szkodliwa rywalizacja

Jeżeli kilka procesów współbieżnie wykorzystuje i modyfikuje te same dane, to wynik działań może zależeć od kolejności w jakiej następował dostęp do danych. Następuje wtedy szkodliwa rywalizacja (*race condition*).

4.2 Problemy synchronizacji

4.2.1 Problem producenta i konsumenta

Problem polega na tym, że producent produkuje ciągle porcje informacji, które powinny być w tej samej kolejności „konsumowane" przez konsumenta. Należy zadbać o to, by producent nie produkował, gdy konsument nie może „skonsumować" porcji danych oraz by konsument nie usiłował konsumować gdy porcja nie została wyprodukowana.

Jednym z rozwiązań takiego problemu jest zastosowanie bufora jako magazynu i nadzorowanie, aby producent nie produkował, gdy bufor jest pełny oraz by konsument nie usiłował konsumować gdy bufor jest pusty.

4.2.2 Problem czytelników i pisarzy

Polega na zsynchronizowaniu dwóch grup cyklicznych procesów konkurujących o dostęp do wspólnej czytelni. Proces czytelnik odczytuje informacje zgromadzone w czytelni i może to robić razem z innymi czytelnikami. Proces pisarz cyklicznie zapisuje informacje i może to robić tylko sam przebywając w czytelni. Czytanie i pisanie trwa skończenie długo. Problem ten jest abstrakcją problemu synchronizacji procesów korzystających ze wspólnej bazy danych.

4.2.3 Problem pięciu filozofów

Polega na zsynchronizowaniu działań pięciu filozofów, którzy siedzą przy okrągłym stole i cyklicznie myślą i jedzą, korzystając z talerza i dwóch widelców. Przed każdym filozofem stoi talerz, a pomiędzy każdymi dwoma talerzami leży jeden widelec. Zakłada się, że jeśli filozof podniósł oba widelce, to w skończonym czasie je odłoży. Rozwiązanie gwarantuje, że każdy filozof, który poczuje się głodny, będzie mógł się w końcu najeść. Ponadto wszyscy filozofowie powinni być traktowani jednakowo. Jest to wzorcowy przykład obrazujący zjawiska zagłodzenia i blokady. Rozwiązanie z możliwością blokady: głodny filozof czeka, aż będzie wolny lewy widelec, podnosi go i czeka na prawy i także go podnosi. Po zjedzeniu odkłada oba widelce. Rozwiązanie z możliwością zagłodzenia: filozof czeka, aż oba widelce będą wolne i wtedy podnosi je jednocześnie. Rozwiązanie poprawne: wykorzystany jest zewnętrzny arbiter (lokaj), który zapewnia, że jednocześnie nie więcej niż czterech (4) filozofów chciałoby jeść, to znaczy konkuruje o widelce. Jeśli pięciu (5) filozofów chciałoby jeść naraz, to lokaj powstrzyma jednego z nich.

4.3 Sekcja krytyczna

Każdy ze współpracujących procesów posiada fragment kodu w którym następuje zmiana wspólnych danych. Jest to sekcja krytyczna procesu. Jeśli jeden z

procesów znajduje się w swojej sekcji krytycznej, inne nie mogą w tym czasie wejść do swoich krytycznych sekcji. Każdy proces musi prosić (w sekcji wejściowej) o pozwolenie na wejście do swojej sekcji krytycznej.

4.3.1 Warunki poprawnego działania

Wzajemne wykluczanie: jeśli proces działa w swej sekcji krytycznej, to żaden inny proces nie działa w swojej.

Postęp: tylko procesy nie wykonujące swoich reszt mogą kandydować do wejścia do sekcji krytycznych i wybór ten nie może być odwlekany w nieskończoność.

Ograniczone czekanie: Musi istnieć graniczna ilość wejść innych procesów do ich sekcji krytycznych po tym, gdy dany proces zgłosił chęć wejścia do swojej sekcji krytycznej i zanim uzyskał na to pozwolenie.

4.4 Mutex

Zamek (z angielskiego *mutex*) jest to obiekt synchronizacji wątków. Definiują się tu dwie operacje: *lock* (zamknij) i *unlock* (otwórz). Wątek próbujący zamknąć zamek już zamknięty zostaje zablokowany. Operacja otwarcia zamku odblokowuje jeden z zablokowanych wątków. Spotyka się też operację trylock (spróbuj zamknąć), która nie blokuje wątku, powiadamiając go o niemożliwości zamknięcia zamku.

4.5 Semafory

Nazwa dobrze oddaje naturę semaforów – jest to mechanizm synchronizacji idealnie pasujący do rozwiązywania problemu sekcji krytycznej, a jego działanie przypomina działanie semafora kolejowego. Wyobraźmy sobie wielotorową magistralę kolejową, która na pewnym odcinku zwęża się do k torów. Pociągi jeżdżące magistralą to procesy. Zwężenie to uogólnienie sekcji krytycznej. Na zwężonym odcinku może znajdować się na raz maksymalnie k pociągów, każdy na oddzielnym torze. Podobnie jak w przypadku sekcji krytycznej, każdy oczekujący pociąg powinien kiedyś przejechać i żaden pociąg nie powinien stać, jeżeli jest wolny tor. Semafor to specjalna zmienna całkowita, początkowo równa k. Specjalna, ponieważ (po zainicjowaniu) można na niej wykonywać tylko dwie operacje:

P – Jeżeli semafor jest równy 0 (semafor jest opuszczony), to proces wykonujący tę operację zostaje wstrzymany, dopóki wartość semafora nie zwiększy się (nie zostanie on podniesiony). Gdy wartość semafora jest dodatnia (semafor jest podniesiony), to zmniejsza się o 1 (pociąg zajmuje jeden z k wolnych torów).

V – Wartość semafora jest zwiększana o 1 (podniesienie semafora). Jeżeli jakiś proces oczekuje na podniesienie semafora, to jest on wznawiany, a semafor natychmiast jest zmniejszany.

Istotną cechą operacji na semaforach jest ich niepodzielność. Jeżeli jeden z procesów wykonuje operację na semaforze, to (z wyjątkiem wstrzymania procesu, gdy semafor jest opuszczony) żaden inny proces nie wykonuje w tym samym czasie operacji na danym semaforze. Aby zapewnić taką niepodzielność, potrzebne jest specjalne wsparcie systemowe lub sprzętowe. Rozwiązanie problemu sekcji krytycznej za pomocą semaforów jest banalnie proste. Wystarczy dla każdej sekcji krytycznej utworzyć odrębny semafor, nadać mu na początku wartość 1, w sekcji wejściowej każdy proces wykonuje na semaforze operację P, a w sekcji wyjściowej operację V. Wymaga to jednak od programisty pewnej dyscypliny: Każdej operacji P musi odpowiadać operacja V i to na tym samym semaforze. Pomyłka może spowodować złe działanie sekcji krytycznej. Jeżeli procesy mogą przebywać na raz więcej niż w jednej sekcji krytycznej, to musimy zwrócić uwagę na możliwość zakleszczenia.

4.6 Monitor

Monitor można traktować jako narzędzie do jednoczesnego definiowania typu danych z dopuszczalnymi na nim operacjami oraz mechanizmami do wzajemnego wykluczania, jeśli danymi ma się manipulować współbieżnie. Składnia monitora polega na hermetyzacji elementów danych i działających na nich procedur w pojedyncze moduły. Interfejs monitora składa się ze zbioru procedur, które operują na danych ukrytych w module (monitorze). Różnica między zwykłym modułem a monitorem jest taka, że monitor nie tylko ochrania wewnętrzne dane przed nieograniczonym dostępem, lecz także synchronizuje wywołania procedur występujących w jego interfejsie. Implementacja zapewnia wzajemne wykluczanie wykonań tych procedur.

4.7 Transakcyjność

Transakcją jest zbiór operacji stanowiących logicznie spójną funkcję. Jest to na przykład ciąg operacji czytania lub pisania zakończonych operacją zatwierdzenia lub zaniechania. Transakcja zaniechana nie powinna pozostawić śladów w danych, które zdążyła już zmienić. Wycofanie transakcji powinno zapewnić odtworzenie danych sprzed transakcji. Spójność danych i ich sprawne odzyskiwanie po np. awarii systemu jest najważniejszą sprawą w bazach danych różnego rodzaju. Jest to też rodzaj synchronizacji danych.

4.7.1 Szeregowanie transakcji

Szeregowanie transakcji polega na takim zaplanowaniu wzajemnego przeplatania się rozkazów z kilku transakcji, aby nie występowały konflikty pisa-

nia/czytania tych samych danych.

4.7.2 Protokół blokowania transakcji

Z każdym obiektem danych kojarzy się zamek, od którego zależy dostęp do danych, np. Jeżeli transakcja dostaje dostęp do obiektu danych w trybie wspólnym, to może czytać ten obiekt, ale nie może go zapisywać

Jeśli dostaje obiekt w trybie wyłącznym, to wolno jej zarówno czytać jak i zapisywać ten obiekt.

Każda transakcja musi zamawiać zamek blokujący obiekt w takim trybie, aby mogła wykonać zaplanowaną operację.

4.8 Odzyskiwanie za pomocą rejestru

Rejestr to zapis w pamięci trwałej, określający wszystkie zmiany w danych wykonywane podczas transakcji. Każdy rekord w rejestrze (logu) zawiera następujące dane:

- nazwa transakcji,
- nazwa jednostki danych,
- stara wartość,
- nowa wartość,
- inne dane dotyczące transakcji, np. zaniechanie

Zanim rozpocznie się wykonywanie transakcji, w rejestrze zapisuje się rekord informujący o rozpoczęciu transakcji. Każdy zapis (przez transakcję) poprzedzony jest zapisem odpowiedniego rekordu w rejestrze. Gdy dochodzi do zatwierdzenia transakcji, w rejestrze zapisuje się rekord zatwierdzenie.

Tworzenie rejestru jest pamięcio- i czasochłonne, ale dla bardzo ważnych danych nie jest to cena wygórowana. Przy rekonstrukcji danych na podstawie rejestru korzysta się z dwóch procedur:

wycofaj - odtwarza wszystkie dane uaktualnbione przez transakcję T, nadając im stare wartości,

przywróć - nadaje nowe wartości wszystkim danym uaktualnionym przez transakcję T.

Transakcja musi być wycofana, jeśli w rejestrze znajduje się rekord rozpoczęcia, a nie ma rekordu zatwierdzenia. Transakcja musi być przywrócona, jeśli w rejestrze jest rekord rozpoczęcia oraz rekord zatwierdzenia dla danej transakcji.

4.8.1 Punkty kontrolne

Odtwarzanie za pomocą rejestru ma pewne wady. Proces przeglądania rejestru jest czasochłonny, większość transakcji zapisanych w rejestrze odbyła się pomyślnie przed awarią, odtwarzanie ich z rejestru jest dublowaniem pracy.

Dla przyspieszenia ewentualnego odtwarzania, system organizuje co jakiś czas tzw. punkty kontrolne, w których wszystkie rekordy pozostające w tej chwili w pamięci operacyjnej są zapisane w pamięci trwałej (na dysku), wszystkie zmienione dane, pozostające w pamięci ulotnej, muszą być zapisane w pamięci trwałej. W rejestrze transakcji zapisuje się rekord *punkt kontrolny*. Po awarii przegląda się rejestr od końca. Po napotkaniu rekordu *punkt kontrolny*, przywracanie rozpoczyna się od pierwszej transakcji po nim.

5 Blokady i zakleszczenia

Blokada, zakleszczenie występuje wtedy, gdy np. w problemie pięciu filozofów jednocześnie każdy z filozofów podniesie jedną pałeczkę i czeka na drugą. Skutek: umrą z głodu W chwili obecnej rzeczywiste systemy operacyjne nie rozwiązują jeszcze problemu zakleszczeń, gdyż są one stosunkowo rzadkie (i mogą być likwidowane przez operatora), ale w niedalekiej przyszłości problem ten stanie przed programistami.

5.1 Warunki wystąpienia zakleszczenia

Jeśli graf przydziału zasobów nie ma cykli, to nie ma zakleszczenia,

Jeśli zasób każdego typu ma tylko jeden egzemplarz, to istnienie cyklu jest warunkiem koniecznym i dostatecznym do wystąpienia blokady,

Jeśli istnieje po kilka egzemplarzy zasobu, to istnienie cyklu jest jedynie warunkiem koniecznym wystąpienia blokady

5.2 Warunki wystąpienia blokady

Do zakleszczenia może dojść, jeśli spełnione są równocześnie warunki:

Wzajemne wykluczanie - co najmniej jeden zasób musi być niepodzielny (w danym czasie może go używać tylko jeden proces)

Przetrzymywanie i oczekiwanie - przynajmniej jeden proces przetrzymuje jakiś zasób, ponieważ czeka na przydział dodatkowego innego zasobu, przetrzymywanego właśnie przez inny proces,

Brak wywłaszczeń - zasób może być zwolniony jedynie z inicjatywy przetrzymującego, np. po zakończeniu procesu,

Czekanie cykliczne - P_1 czeka na zasób przetrzymywany przez P_2, P_2 czeka na oddanie przez $P_3 \cdots P_n$ czeka na zwolnienie zasobu przez P_1

5.3 Metody eliminacji

Stosowanie protokołu gwarantującego, że system nigdy nie wejdzie w stan zakleszczenia,

Pozwolenie na występowanie zakleszczeń i podjęcie stosownych działań po takim fakcie,

Zlekceważenie problemu (np. UNIX) - założenie, że zakleszczenia nie pojawiają się.

Praktycznie w takich systemach zakleszczenia pojawiają się rzadko (np. raz do roku), więc raczej się nie inwestuje w kosztowne mechanizmy do unikania zakleszczeń lub ich likwidacji, a wykonuje się to ręcznie.

5.4 Metody zapobiegania

Trzeba zapewnić, aby nie wystąpił co najmniej jeden z warunków koniecznych:

Wzajemne wykluczanie - nie można uniknąć tego warunku, gdyż pewne zasoby są z natury niepodzielne, np. drukarki,

Przetrzymywanie i oczekiwanie - trzeba zagwarantować, że gdy proces zamawia jakiś zasób, to nie przetrzymywuje innych zasobów (proces otrzymuje wszystkie zasoby przed rozpoczęciem działania, lub może zamówić zasób, jeśli wcześniej oddał wszystkie pozostałe)

Brak wywłaszczeń - trzeba dopuścić, aby nie tylko proces mógł zwolnić zasoby -jeśli proces nie może dostać żądanego zasobu, to traci pozostałe i czeka na nie, - jeśli proces zamawia jakieś zasoby, to się sprawdza czy są dostępne: jeśli tak, to się je przydziela, jeśli zasób jest trzymany przez inny proces, który czeka na jakieś zasoby, to mu się zasoby odbiera i daje zamawiającemu. W przeciwnym wypadku proces czeka i może utracić swoje zasoby.

Czekanie cykliczne - aby nie wystąpiło, trzeba zasobom tego samego typu nadać liczbowe identyfikatory (np. taśmy - 2, drukarki - 5) i trzeba dbać, aby proces zamawiał zasoby w rosnącym porządku numeracji, ewentualnie aby zwalniał zasoby o numeracji wyższej od zamawianego.

Wadą jest słabe wykorzystanie zasobów i możliwość głodzenia procesów.

5.5 Unikanie zakleszczeń

Każdy proces deklaruje maksymalną liczbę zasobów danego typu. Stan przydziału zasobów jest określony przez liczbę dostępnych i przydzielonych zasobów oraz przez maksymalne zapotrzebowanie na zasoby. Algorytm unikania zakleszczeń sprawdza stan przydziału zasobów i blokuje niektóre przydziały, aby nie doszło do czekania cyklicznego. Jeśli porządek przydzielania zasobów procesom jest taki, że nie ma możliwości wystąpienia zakleszczeń, to system jest w **stanie bezpiecznym**.

5.6 Stan bezpieczny

Jeżeli dla każdego procesu P_i (spośród $P_1 \cdots P_n$) jego zapotrzebowanie na zasoby może być zaspokojone przez aktualnie dostępne zasoby, oraz przez zasoby używane przez wszystkie procesy P_j, gdzie $j < i$, to taki ciąg procesów jest bezpieczny. Jeżeli P_i nie ma danej chwili dostępnych wszystkich zasobów, to poczeka aż procesy P_j skończą działanie i oddadzą swoje zasoby P_i Po otrzymaniu wszystkich zasobów proces Pi może dokończyć pracę i zwolnić zasoby Wtedy proces P_{i+1} może otrzymać zasoby (jeśli na nie czekał) itd.

5.7 Wykrywanie zakleszczeń

Jeśli nie unikamy zakleszczeń to w systemie muszą istnieć:

1. Algorytmy wykrywania zakleszczenia - tworzy się graf oczekiwania i okresowo wykonuje algorytm poszukiwania cykli w tym grafie
2. Algorytmy likwidowania zakleszczenia - usunięcie wszystkich zakleszczonych procesów lub usuwanie procesów pojedynczo aż do wyeliminowania cyklu zakleszczenia.

Kryteria wyboru ofiary w algorytmie likwidowania zakleszczenia:

1. Priorytet
2. Czas wykonywania i pozostały do zakończenia
3. Liczba i typ zasobów przetrzymywanych przez proces
4. Liczba procesów, które trzeba będzie przerwać
5. Proces interakcyjny, czy wsadowy

5.8 Wywłaszczanie

Wybierając ofiarę do wywłaszczenia trzeba mieć na uwadze minimalizację kosztów - ile zasobów odzyskamy, a ile pracy procesu stracimy

Wycofanie (o ile to możliwe) procesu do bezpiecznego punktu, od którego może wznowić działanie

Głodzenie to w tym wypadku wywłaszczanie ciągle tego samego procesu. Algorytm powinien więc zapewniać, że proces będzie ofiarą tylko skończoną ilość razy.

5.9 Algorytm bankiera

Każdy proces wchodzący do systemu musi zadeklarować maksymalną liczbę używanych egzemplarzy każdego zasobu, a liczba ta musi być nie większa od całkowitych zasobów w systemie. System decyduje, czy przydział tych zasobów pozostawi stan bezpieczny Jeśli tak, to przydziela. Jeśli nie - to proces musi poczekać na zwolnienie zasobów przez inne procesy. Jeśli proces uzyska potrzebne zasoby, to musi je zwrócić w skończonym czasie.

6 Zarządzanie pamięcią

Przed wykonaniem program musi być pobrany z dysku i załadowany do pamięci. Tam działa jako proces. Podczas wykonywania, proces pobiera rozkazy i dane z pamięci. Większość systemów pozwala procesowi użytkownika przebywać w dowolnej części pamięci fizycznej. System musi uwzględniać to, że program „nie wie" pod jakim adresem pamięci będzie umieszczony. System musi zawierać mechanizmy „tłumaczące" adresy w programie na rzeczywiste adresy w pamięci fizycznej. System musi też gospodarować wolną pamięcią, racjonalnie ładując kolejne programy w wolne miejsca pamięci

6.1 Powiązanie programu z adresami w pamięci

Powiązanie programu z adresem w pamięci może być dokonane w trzech fazach:

Czas kompilacji - jeśli podczas kompilacji wiadomo pod jakim adresem pamięci program będzie przebywał, to tworzy się kod bezwzględny, np programy .com w DOS-ie. Aby zmienić położenie takiego programu w pamięci, trzeba go powtórnie przekompilować. Jeśli kod programu ma być przemieszczalny, to adresy muszą być określone w sposób względny, np w odległościach od początku modułu.

Czas ładowania - jeśli podczas kompilacji nie określono rzeczywistych adresów pamięci, to program ładujący wylicza je na podstawie adresów względnych.

Czas wykonania - jeśli proces może ulegać przemieszczeniu w pamięci podczas wykonywania, to muszą istnieć mechanizmy wyliczania w dowolnym momencie rzeczywistych adresów.

6.2 Dołączanie dynamiczne

Ładowanie dynamiczne - podprogram nie jest wprowadzany do pamięci dopóty, dopóki nie zostanie wywołany. Do pamięci wprowadza się jedynie program główny, a potem sukcesywnie potrzebne moduły. Dzięki temu oszczędza się miejsce w pamięci nie ładując nie potrzebnie wielkich modułów, np obsługi błędów.

Konsolidacja dynamiczna - bez tej właściwości wszystkie programy muszą mieć dołączone kopie bibliotek, w tym systemowych. Powoduje to marnotrawstwo dysku i pamięci. Biblioteki dynamicznie linkowane są sprowadzane do pamięci w momencie ich wywołania i tam mogą służyć nawet kilku programom. Dodatkowa zaleta - aktualizacja biblioteki nie wymaga zazwyczaj przekompilowania programu.

6.3 Nakładki

Nakładki są potrzebne jeśli proces jest większy niż ilość dostępnej pamięci. Program nakładkowy nie potrzebuje specjalnego wsparcia ze strony systemu operacyjnego, ale wymaga starannego programowania, ze znajomością rzeczy.

6.4 Adresy fizyczne i logiczne

Adres fizyczny - jest to adres oglądany przez jednostkę pamięci (umieszczony w jej rejestrze adresowym)

Adres logiczny - jest to adres wygenerowany przez procesor

Odwzorowanie adresów logicznych na fizyczne realizowane jest sprzętowo przez jednostkę zarządzania pamięcią (MMU). W MMU przeliczanie adresu odbywa się najczęściej poprzez dodanie do adresu z procesora wartości rejestru przemieszczenia. Program użytkownika działa wyłącznie na logicznych adresach

6.5 Wymiana, czyli *swapping*

Wymiana (*swapping*) jest to tymczasowe odesłanie procesu do pamięci pomocniczej (na dysk) i przepisanie z powrotem w celu kontynuowania działania. Zazwyczaj po wymianie proces powraca na to samo miejsce w pamięci, chyba że

są zapewnione mechanizmy przeliczania adresów. Warunkiem szybkiej wymiany jest używanie dysku o krótkim czasie dostępu i o pojemności zapewniającej pomieszczenie obrazów pamięci wszystkich użytkowników. W planowaniu priorytetowym stosuje się czasem wymianę poprzez *wytaczanie* procesu, gdy nadchodzi proces o wyższym priorytecie i *wtaczanie* z powrotem do pamięci, gdy procesy wysoko-priorytetowy skończył działanie

6.6 Przydział ciągły

Pamięć operacyjna zajęta jest przez system operacyjny - umieszczony zazwyczaj w tej części pamięci, gdzie wektor przerwań - jest to najczęściej **pamięć dolna** oraz procesy użytkownika, umieszczane zazwyczaj w **pamięci górnej**.

W celu ochrony obszaru pamięci zajętego przez system, a także procesów użytkownika przed wzajemną ingerencją, wykorzystuje się rejestr przemieszczenia i rejestr graniczny

rejestr przemieszczenia - wartość najmniejszego adresu fizycznego dla danego procesu (*offset*)

rejestr graniczny - maksymalny adres logiczny procesu

6.7 Kryteria wyboru wolnego obszaru

Dziura jest to ciągły obszar niezajętej pamięci

6.7.1 Pierwsze dopasowanie

Pierwsze dopasowanie - system przydziela pierwszą dziurę o wystarczającej wielkości. Szukanie rozpoczyna się od początku wykazu dziur lub od miejsca w którym zakończono ostatnie szukanie.

6.7.2 Najlepsze dopasowanie

Najlepsze dopasowanie - przydziela się najmniejszą z dostatecznie dużych dziur (najmniejsza pozostałość po przydziale)

6.7.3 Najgorsze dopasowanie

Najgorsze dopasowanie - przydziela się największą dziurę. Czasami duża pozostałość po takim przydziale jest bardziej przydatna niż malutkie fragmenty po najlepszym dopasowaniu. Ciekawe i oryginalne podejście do zagadnienia, ale

praktyka wykazała, że dwie pozostałe metody są lepsze pod względem czasu działania i wykorzystania pamięci.

6.8 Fragmentacja pamięci

6.8.1 Fragmentacja zewnętrzna

Fragmentacja zewnętrzna występuje, gdy suma wolnych obszarów pamięci wystarcza na spełnienie zamówienia, ale nie tworzą one spójnego obszaru. Fragmentację zewnętrzną można zmniejszyć poprzez takie upakowanie procesów, aby cała wolna pamięc znalazła się w jednym dużym bloku. Jest to możliwe tylko wtedy, gdy ustalanie adresów jest wykonywane dynamicznie podczas działania procesu.

Przetasowań procesów nie można robić podczas operacji we/wy.

6.8.2 Fragmentacja wewnętrzna

Fragmentacja wewnętrzna występuje, jeśli po przydzieleniu pamięci do procesu pozostałby wolny obszar wielkości kilku bajtów, to przydziela się go też do procesu, ale stanowi on „nieużytek" - nie będzie wykorzystany (ale zmniejszy się tablica „dziur")

6.9 Stronicowanie

Stronicowanie pomaga w racjonalnym wykorzystaniu wolnych miejsc w pamięci Pamięć fizyczną dzieli się na bloki o stałej długości (ramki) o długości 2^n (512 B do 16 MB). Pamięć logiczna podzielona jest na strony o tym samym rozmiarze. Wolną pamięć obrazuje lista wolnych ramek. Proces o wielkości N stron jest ładowany w N ramek (niekoniecznie kolejnych). Tablica stron odwzorowuje adresy logiczne na fizyczne. W ten sposób eliminuje się fragmentację zewnętrzną, ale występuje fragmentacja wewnętrzna (zaokrąglenie w górę wielkości procesu do wielokrotności rozmiaru ramki).

Każdy adres wygenerowany przez procesor dzieli się na dwie części: numer strony i odległość na stronie. Numer jest używany jako indeks w tablicy stron, która zawiera adresy bazowe wszystkich stron w pamięci operacyjnej. Łącząc adres bazowy z odległością na stronie uzyskuje się fizyczny adres w pamięci.

Jeżeli rozmiar strony jest potęgą 2 oraz:

- rozmiar przestrzeni adresowej wynosi 2^m
- rozmiar strony wynosi 2^n,

to $m-n$ bardziej znaczących bitów adresu logicznego wskazuje nr strony (2^{m-n}), n mniej znaczących bitów wskazuje odległość na stronie.

6.9.1 Tablica stron

Jest przechowywana w pamięci operacyjnej. Jej położenie wskazuje rejestr bazowy tablicy stron. Rozmiar tablicy stron jest przechowywany w rejestrze długości tablicy stron. Określa on na największy dopuszczalny adres. Przy korzystaniu z tablicy stron, dostęp do pamięci wymaga dwukrotnego dostępu do pamięci. W celu przyspieszenia dostępu do pamięci stosuje się rozwiązanie sprzętowe - małą, szybką pamięć podręczną zwaną **rejestrami asocjacyjnymi** lub **buforami translacji adresów stron**. Bufory te zwierają 8 do 2048 pozycji. Jeśli dany numer strony nie znajduje się w buforach, to przeszukiwana jest cała tablica stron. Przy dobrze skonstruowanym algorytmie, w buforach translacji znajduje się 80 do 98 % potrzebnych numerów stron.

6.9.2 Stronicowanie wielopoziomowe

Nowoczesne systemy zezwalają na na stosowanie bardzo wielkich przestrzeni adresów logicznych (2^{32} do 2^{64}) W takim przypadku tablica stron może zawierać nawet milion wpisów. Jeśli każdy wpis to 4B, rozmiar tablicy wyniesie 4MB, na każdy proces. Tablice takie mogą być większe niż same procesy. Celowy jest więc podział na mniejsze tablice. Przy 32-bitowej przestrzeni adresowej 20-bitowy adres strony i 12-bitową odległość na stronie można zastąpić przez 10-bitowy adres strony, 10-bitową odległość na tej zewnętrznej stronie i 12-bitową odległość wewnętrzną. Każdy poziom jest zapisywany jako oddzielna tablica, więc przekształcenie adresu logicznego w fizyczny może wymagać czterech dostępów do pamięci.

6.9.3 Odwrócona tablica stron

Odwrócona tablica stron ma po jednej pozycji dla każdej ramki w pamięci fizycznej Każda pozycja zawiera numer procesu posiadającego ramkę oraz adres wirtualny strony przechowywanej w ramce rzeczywistej pamięci. W systemie istnieje tylko jedna tablica stron - ogranicza to zajętość pamięci, ale zwiększa czas przeszukiwania (trzeba przeszukać całą tablicę) Stosowanie tablic hashowania ogranicza przeszukiwanie do co najwyżej kilku wpisów.

6.9.4 Strony dzielone

20 użytkowników korzysta równocześnie z edytora tekstu. W pamięci musi się znaleźć 20 bloków danych i 20 kopii kodu edytora. Jeśli kod programu nie modyfikuje sam siebie w czasie działania (jest wznawialny) to można zastosować mechanizm stron dzielonych. Wznawialny kod programu jest widziany przez

wszystkie procesy pod tą samą lokacją. Każdy proces dysponuje więc własnym obszarem danych i jednym wspólnym kodem programu Mechanizm ten może być też stosowany przy innych intensywnie używanych programach (kompilatory, systemy okien, bazy danych).

6.9.5 Stronicowanie dwupoziomowe

W stronicowaniu dwupoziomowym istnieje dodatkowo **katalog stron**, zawierający adresy tablic stron. Liniowy adres jest 16-bitową liczbą:

- 20 najstarszych bitów - numer strony
- 10b - wskaźnik do katalogu stron,
- 10b - wskaźnik do tablicy stron,
- 12 najmłodszych bitów - odległość na stronie

6.10 Segmentacja

Segmentacja jest to mechanizm naśladujący postrzeganie pamięci tak jak użytkownik. Przestrzeń adresów logicznych jest zbiorem segmentów. Każdy segment ma nazwę i długość. Użytkownik określa więc każdy adres poprzez nazwę segmentu i odległość. Kompilator automatycznie tworzy segmenty tworzące program wynikowy. Najczęściej oddzielnymi segmentami są:

- program główny,
- procedury,
- funkcje,
- zmienne lokalne,
- zmienne globalne,
- stos wywołań procedur (parametry i adresy powrotu)
- bloki wspólne (common) $\cdots$

Adres logiczny składa się z dwóch części - numeru segmentu i odległości w segmencie Tablica segmentów zawiera pary danych:

- baza (fizyczny adres początku segmentu w pamięci)
- granica (długość segmentu)

Rejestr bazowy tablicy segmentów jest to adres tablicy segmentów w pamięci. Ponieważ programy mogą mieć bardzo różniącą się liczbę segmentów, stosuje się też **rejestr długości tablicy segmentów**, który służy do sprawdzenia czy podany numer segmentu jest poprawny (¡ RDTS)

6.10.1 Segmentacja ze stronicowaniem

W systemach opartych na architekturze i386 występuje segmentacja ze stronicowaniem z dwupoziomowym schematem stronicowania:

- maksymalna liczba segmentów w procesie: 16K
- każdy segment mniejszy niż 4 GB
- rozmiar strony 4 kB
- przestrzeń adresowa ma dwie strefy po co najwyżej 8K segmentów.

Prywatne segmenty procesów przechowywane są w tablicy lokalnych deskryptorów, natomiast wspólne segmenty procesów przechowywane są w globalnej tablicy deskryptorów.

Selektor jest 16-bitową liczbą zawierającą 13b numer segmentu, 1b czy segment jest w lokalnej czy globalnej tablicy deskryptorów, oraz 2b ochrony. Każdy adres logiczny jest parą: selektor, odległość. Procesor ma 6 rejestrów segmentów (do adresowania 6 segmentów) oraz 6 rejestrów mikroprogramowych do przechowywania pozycji z lokalnej i globalnej tablicy deskryptorów. Sprawdzanie adresu: rejestr wyboru wskazuje na pozycję w lokalnej lub globalnej tablicy - na podstawie adresu początku segmentu i jego długości tworzy się adres liniowy (sprawdzenie poprawności). Jeśli jest on poprawny, to do bazy dodaje się odległość.

6.11 Ochrona pamięci

Aby chronić pamięć stosuje się **bity ochrony**, przypisane do każdej ramki. Można w ten sposób zaznaczyć strony tylko do czytania, do czytania i pisania i do wykonywania. **Bit poprawności** jest ustawiony, jeśli dana strona jest w przestrzeni adresowej procesu (strona jest „legalna” dla procesu). Jeśli strona jest poza przestrzenią adresową procesu, ma ustawiony znacznik „nielegalna”.

7 Pamięć wirtualna

Pamięć wirtualna odseparowuje pamięć logiczną od jej fizycznej realizacji. Można ją zaimplementować jako stronicowanie na żądanie lub segmentację na żądanie

Zalety korzystania z pamięci wirtualnej:

Program nie jest ograniczony wielkością pamięci fizycznej - można pisać ogromne programy bez specjalnych „sztuczek” programistycznych

Każdy program zajmuje w pamięci mniej miejsca niż program „kompletny". Można więc w tym samym czasie wykonywać więcej zadań, polepszając wykorzystanie procesora

Maleje liczba operacji wejścia-wyjścia koniecznych do załadowania programów do pamięci oraz do realizacji wymiany - programy powinny więc wykonywać się szybciej

Nie ma konieczności robienia nakładek przy małej pamięci operacyjnej

7.1 Stronicowanie na żądanie

Procesy przebywają w pamięci pomocniczej (na dysku). Dla wykonania sprowadza się proces do pamięci, ale nie cały, tylko te strony, które są potrzebne (leniwa wymiana). Typowanie (zgadywanie) potrzebnych stron odbywa się podczas poprzedniego pobytu procesu w pamięci. Jeśli proces odwołuje się do strony, której nie ma w pamięci, to system sprawdza, czy odwołanie do pamięci było dozwolone czy nie (bit poprawności). Jeśli było poprawne, sprowadza stronę do pamięci, modyfikuje tablicę stron i wznawia działanie procesu.

7.1.1 Stronicowanie na żądanie, a wymiana

Proces jest ciągiem stron

Wymiana dotyczy całego procesu (wszystkich stron)

Procedura stronicująca dotyczy poszczególnych stron procesu

Procedura stronicująca zgaduje, jakie strony będą w użyciu i tyko je ładuje do pamięci

Nigdy nie dokonuje się wymiana całego procesu

7.1.2 Zastępowanie stron

Założenie, że tylko część stron każdego procesu jest potrzebna w pamięci może doprowadzić do nadprzydziału (nadmiar procesów w pamięci i absolutny brak wolnych ramek). Aby nie blokować procesu potrzebującego kolejnej ramki, stosuje się zastępowanie stron. Zastępowanie stron jest podstawą stronicowania na żądanie. Praktycznie każdy proces wykonuje się z użyciem mniejszej ilości ramek niż by wynikało z wielkości procesu. Mechanizm działa efektywnie przy dobrze opracowanych algorytmach przydziału ramek i algorytmach zastępowania stron.

Uruchamia się algorytm typowania ramki-ofiary

- stronę-ofiarę zapisuje się na dysku,
- aktualizuje się tablicę wolnych ramek,

- wczytuje się potrzebną stronę do zwolnionej ramki
- aktualizuje się tablicę stron
- wznawia się działanie procesu

7.1.3 Algorytmy zastępowania stron

Algorytm FIFO (*first in, first out*) - o każdej ze stron zapamiętuje się informację, kiedy ona została sprowadzona do pamięci i zastępuje się „najstarszą" stronę.

Algorytm LRU (*least recently used*) - zastępowanie stron, które były najdawniej używane. Typowanie najstarszych odbywa się poprzez licznik (w tablicy stron jest rejestr czasu ostatniego używania strony) lub stos (przy każdym odwołaniu do strony, jej numer jest wyjmowany i umieszczany na szczycie stosu. Algorytmy bazujące na metodzie LRU:

z bitami odniesienia (po odwołaniu do strony, znacznik przyjmuje wartość 1)

dodatkowe bity odwołań (co stały czas ustawianie kolejnych bitów rotacyjnie)

druga szansa (jeśli bit odniesienia = 1 to zeruje się go, zmienia czas na bieżący i przegląda kolejne strony - FIFO. Jeśli bit = 0 to się stronę wymienia)

ulepszony algorytm drugiej szansy: wykorzystuje się dwa bity: bit odniesienia i bit modyfikacji, jako parę. Powstają cztery klasy stron:

(0,0) - nie używana ostatnio i nie zmieniana (najlepsza ofiara)

(0,1) - nie używana ostatnio, ale zmieniana (gorsza, bo trzeba ją zapisać)

(1,0) - Używana ostatnio, ale nie zmieniana (prawdopodobnie za chwilę będzie znów używana)

(1,1) - używana ostatnio i zmieniona (chyba będzie używana niedługo, a poza tym trzeba ją zapisać - najgorsza kandydatka na ofiarę)

Zastępujemy pierwszą napotkaną stronę z najniższej klasy

7.2 Przydział ramek

Każdemu procesowi system mysi przydzielić pulę ramek pamięci potrzebnych do jego pracy. Trzy najpopularniejsze algorytmy przydziału:

równy (każdy proces dostaje tyle samo ramek) np. 50 ramek i 5 procesów, to każdy dostaje po 10

proporcjonalny (liczba przydzielonych ramek proporcjonalna do wielkości procesu)

priorytetowy (liczba przydzielonych ramek zależy tylko od priorytetu procesu, albo od priorytetu i wielkości)

7.3 Zastępowanie ramek

Zastępowanie lokalne - proces może zastępować ramki wyłącznie z puli tych, które dostał przy przydziale

Zastępowanie globalne - Proces może korzystać z puli wszystkich wolnych ramek, nawet jeżeli są wstępnie przydzielone innemu procesowi. Proces może zabrać ramkę drugiemu.

Praktyka wykazała, że zastępowanie globalne daje lepszą przepustowość systemu.

7.4 Szamotanie

Jeśli proces nie ma wystarczającej liczby ramek, to w pewnym momencie musi wymienić stronę, która będzie potrzebna w niedługim czasie. W konsekwencji, kolejne braki stron będą występowały bardzo często. Taki proces „szamoce się", spędzając więcej czasu na stronicowaniu niż na wykonaniu. Zmniejsza się wykorzystanie procesora i koło się zamyka. Jak powstrzymać szamotanie lub do niego nie dopuścić?

- stosować metodę lokalnego lub priorytetowego przydziału stron
- stosować mechanizmy zmniejszenia wieloprogramowości przy szamotaniu
- zapewnić dostawę wystarczającej ilości ramek.

7.5 Problem operacji I/O

Proces zamawia operację we-wy i ustawia się w kolejce do urządzenia, procesor przydziela się innym procesom. Zaczynają występować braki stron. W wyniku algorytmu globalnego zastępowania stron zostaje wymieniona strona zawierająca bufor I/O procesu czekającego na operację I/O. Zaczyna się operacja we-wy i nadpisuje dane innego procesu!

Zapobieganie:

- zakaz wykonywania operacji I/O wprost do pamięci użytkownika
- blokowanie stron w pamięci - strony czekające lub realizujące operację I/O nie mogą być zastępowane

8 Zakończenie

8.1 Słowo na zakończenie

Plik został napisany oraz skonwertowany do PDF przy pomocy systemu LaTeX w systemie Debian GNU/Linux. Z powodu mojego wrodzonego lenistwa nie ma sekcji o zarządzaniu plikami i miejscem na dysku oraz o urządzeniach I/O. Egzamin mam już za sobą, więc raczej nie będę tego uaktualniał, a jeżeli ktoś chce mieć możliwość dalszego rozwijania tematu, wystarczy wysłać do mnie maila na *m.ciesla@gmail.com* - mogę udostępnić plik źródłowy LaTeXa. Powodzenia!

8.2 Przydatne linki

Ewolucja systemu MS-DOS - informacje na temat rozwoju pierwszego ogólnodostępnego SO, jakim był MS-DOS

http://republika.pl/p_bogus/software/msdos/msdos.htm

Zbigniew S. Szewczak - Podstawy Systemów Operacyjnych - wykłady z SO na Uniwersytecie w Toruniu

http://www.mat.uni.torun.pl/žssz/PSO2003/

MSDN - Microsoft Developer Network - masa dokumentacji do systemu operacyjnego *MS Windows* po angielsku

http://msdn.microsoft.com

The Linux Documentation Project - projekt dokumentowania systemu Linux (po angielsku)

http://www.tldp.org

Linux Programmers' Guide PL - polskie tłumaczenie podręcznika programisty Linuksa

http://lpg.foo-baz.com/

Polish Debian Documentation Project - projekt spolszczania dokumentacji systemu operacyjnego Debian GNU/Linux

http://debian.linux.org.pl/

JTZ - Jak To Zrobić - projekt spolszczania Linux-HOWTO

http://www.jtz.org.pl/