Relacja dziecko-rodzic - każdy proces oprócz inita ma rodzica, inaczej mówiąc każdy proces jest subprocesem innego.
Identyfikator procesu (PID) - unikatowy index procesu, maksymalnie może mieć wartość $2^{15} = 32768$ lub $2^{22} = 4194304$.
Identyfikator grupy (PGID) - unikatowy index grupy do której należy proces, ułatwia to budowanie drzewa procesów oraz ułatwia wysyłanie sygnałów, jak i samą kuminkację między nimi.
Wątki jądro - operują w kernelspace, są tworzone przez inne uprzywilejowane wątki.
Proces zombie - proces który został "zabity"ale jeszcze nie jest "pogrzebany", inaczej zwolniony proces który pobiera zasoby systemowe. Aby pogrzebać zombie należy w rodzicu zrobić waitpid lub wait albo poczekać aż sam proces się zterminuje i wtedy init go pogrzebie.
Process sierota - proces którego rodzic umarł ale jeszcze on działa. Sierota jest "przepina"pod najsilniejszego subprocesa w grupie lub do inita.
Segmety programu - definiowane w elfie, zawierajace 1 lub więcej ładowalnych sekcji programu, mają swoje atrybuty i z reguły są ciągłe
pliki odwzorowane w pamięć - segment w pamięci który ma bezpośrednie mapowanie w nazej pamięci (bajt do bajta)
Zasoby plikopodobne - zasób mający deskryptor pliku, ale nie będący zwykłym plikiem (np. socket sieciowy)
Gniazdo unix (socket) - pseudoplik służący do komunikacji sieciowej. Składa się z "2 końców", czyli urządzenia wysyłającego i otrzymującego dane, każdy socket musi mieć zdefiniowane: jakim typem połączenia się posługuje (TCP/UDP) oraz domena w jakiej działa (rodzina protokołów jaką będzie się posługiwał)
Potok (pipe) FIFO - mechanizm komunikacji międzyprocesowej. Najczęściej łączy się stdin programu A z stdout B.
Stany procesu: *Stoppped* - wstrzymany, może zostać wznowiony (np. przez syscall). *Zombie* - zwolniony proces który pobiera zasoby systemowe. *Running*, który się dzieli: Ready - załadowany do pamięci czeka na na wykonanie oraz Executing - proces który obecnie jest wykonywany *Uninterruptible (sen nieprzerywalny)* - proces jest zablokowany i nie reaguje na żadne sygnały. *Interruptible (sen przerywalny)* - proces jest zablokowany, ale może oczekiwać na koniec jakiejś operacji, albo na jakiś sygnał
Zablokowanie sygnału - niedostarczenie sygnału do procesu aż do momentu odblokowania go przez SO. Dzięki temu nie tracimy informacji o sygnale, a jednocześnie nie musimy na niego reagować w niewygodnym dla nas momencie.
Zignorowanie sygnału - sytuacja kiedy proces nie ma zdefiniowanego zachowania dla danego sygnału.
Sygnału SIGKILL nie da się zablokować. Ignorowanie kończy się zabiciem procesu
Elementy dziedziczone po forku: otwarte deskryptory plików, Real User ID, Real Group ID, Effective User ID, Effective Group ID, Suplementary Group IDs, Process Group ID, Session ID, Controlling Terminal, set-user-ID oraz set-group-ID flags, aktualnie używany plik/folder, root directory, mask do tworzenia plików, maske sygnałową, close-on-exec flags do otwatych plików, środowisko, podłączone współdzielone odcinki pamięci, mapowanie pamięci, ograniczenia zasobów.
Elementy dziedziczone po exec: process ID, parent ID, real user ID, real group ID, suplementary group IDs, process group ID, session ID, controlling terminal, time left until alarm clock, aktualnie używany plik/folder, root directory, maska do tworzenia plików, blokady plików, mask sygnałowa procesu, ograniczenia zasobów, deskrypoty otwartych plików.
Sygnał oczekujący - to sygnały, których dostarczenie jest wstrzymane do momentu wyjścia przez proces z nieprzerywalnego snu
Osierocenie - parent process terminuje się przed dzieciorem
Zadanie drugoplanowe - proces/grupa procesów działająca w tle, nie mają dostępu do operacji read/write do tty
Lider sesji - program zarządzający grupą procesów, czyli sesją (zazwyczaj jest to shell - bash/zsh). Kooperuje z kernelem za pomocą sygnałów, system calli
Terminal sterujący - urządzenie znakowe, odpowiada za efekt wpisywanych przez nas komend do emulatora terminala, porozumiewa się z procesami, zarządza uprawnieniami (r/w) mówi które są pierwszoplanowe/drugoplanowe.
Tryb kanoniczny terminala - czyta całe ciągi znaków i je interpretuje.
Tryb niekanoniczny terminala - czyta pojedyncze znaki i ich interpretuje.
Pseudoterminal - para urządzeń z komunikacją w dwie strony - master i slave.
Master - jest od strony użytkownika i przekazuje wszystko do slave
Slave - służy jako terminal kontrolny, do kórego mogą przyłączać się procesy, pisać, czytać etc.
Script - podczas sesji terminala loguje wszystkie znaki z input i output queue i zapisuje do pliku
Nieużytek - nieużywany fragment reprezentacji katalogu - rozmiar wpisu, po którym występuje nieużytek jest większy niż nazwa pliku.
Kompaktowanie - Operacja, która zmniejsza rozmiar katalogu - usuwane są nieużytki, "uklepujemy" katalog.
Ścieżka bezwzględna - Ścieżka zaczynająca się od katalogu /
i-node - struktura przechowująca metadane
Dowiązania - Dowiązania są zawsze tworzone do i-node'a, który nie jest dzielony między różnymi systemami plików. W dwóch systemach plików możemy mieć dwa pliki o tym samym numerze i-node.
Memory-mapped files - Tworzymy region w pamięci i wklejamy tam content jakiegoś pliku albo jego część
Anonymous mappings - pusty szablon inicjalizowany 0
Odwzorowanie prywatne - niewidoczne dla innych procesów. Zazwyczaj przy fork-u jest Copy in write
Odwzorowanie dzielone - widoczne dla innych procesów.
Private mapping file - głównym zastosowaniem jest zainicjalizowanie obszaru pamięci z zawartości pliku, np. inicjalizacja sekcji text lub data procesu z pliku wykonywalnego lub biblioteki współdzielonej.
Private mapping anonymous - alokacja nowej pamięci (pustej, tzn. wypełnionej zerami) dla procesu, np. używając malloc(3), który wykorzystuje mmap(2).

Shared mapping file - przy I/O dla plików dostarczana jest alternatywa dla używania read(2) oraz write(2) lub umożliwienie szybkiej komunikacji międzyprocesowej (IPC) dla procesów niepowiązanych ze sobą
Shared mapping anonymous - umożliwienie szybkiej komunikacji międzyprocesowej (IPC) dla procesów powiązanych ze sobą
Zbiór roboczy - zbiór adresów na którym aktualnie pracuje program
Zbiór rezydentny - część address space procesu która aktualnie znajduje się w pamięci np. skasowane pliki, fork() itd. Więc pamięć zostanie policzona wielokrotnie
Minor page fault - Kiedy kernel szuka strony o którą prosi proces, ale strona jest juz w pamięci, bo inny proces już ją sprowadził
Major page fault - Kiedy strona musi zostać sprowadzona z dysku
«mm_struct::pgd» - pointer na tablicę stron, tablica stron jest tworzona na podstawie opisu segmentów z mmap
«mm_struct::mmap» - pointer na listę segmentów pamięci procesu, każdy segment jest typu vm_area_struct
«SEGV_MAPERR» - Kiedy odwołamy się do niezmapowanego segmentu pamięci
«SEGV_ACCERR» - Kiedy nie mamy uprawnień żeby coś zrobić np. nadpisać .text
Kopiowanie przy zapisie - technika która mówi o tym, żeby prywatne mappingi były współdzielone z innymi, ale kopiowane w chwili kiedy próbujemy wykonac do niej zapis
Struktura «vm_area_struct»: *«vm_prot»* - uprawnienia dostępu r/w; *«vm_flags»* - różne flagi dotyczące np. czy mapping jest shared czy private; *«vm_start/vm_end»* - początek/koniec mappingu; *«vm_nex/vm_prev»* - następny/poprzedni mapping; *pgd* - pointer na tablicę stron
Stronnicowanie na żądanie - leniwe sprowadzanie stron z dysku/cache
Fragmentacja wewnętrzna - kiedy mamy duży wolny blok, ale po zaalokowaniu w nim pamięci requestowanej, duża część tego bloku jest marnowana (bo np. był request 0x20B, a blok miał 0x30B)
Fragmentacja zewnętrza - kiedy mamy wolne bloki gotowe do alokacji, ale nie możemy ich wziąć do zaalokownia requestu programu (bo np. był za duży)
Polityka ramps - programy akumulują struktury monotonicznie przez jakiś czas i trzymają, aż nie będzie potrzebna (zazwyczaj szybko zwalniają)
Polityka peaks - program gwałtownie alokuje dużo pamięci, wykonuje każdą(?) czynność, a następnie zwalnia pamięć. Zostaje parę bloków które reprezentują rezultat pracy
Polityka plateaus - szybko alokują, ale nie zwalniają (czasem aż do końca programu)
first-fit - zobacz liste free bloków i weź pierwszy który pasuje
next-fit - jak first-fit, ale zacznij szukać od miejsca, gdzie poprzednio skończyliśmy
best-fit - znajdź taki który zmarnuje najmniej miejsca
gorliwe złączanie - próbuje od razu połączyć wolne blok "za" i "przed" nim
Protokoły warstwy łącza - Warstwa sieciowa wysyła datagramy które wędrują przez wiele routerów będących pomiędzy źródłem a celem. Aby wysyłać pakiety z jednego wierzchołka (hosta lub routera) do drugiego, warstwa sieciowa polega na warstwie łącza i jej protokołach. Warstwa sieciowa w każdym napotkanym wierzchołku przekazuje datagram warstwie łącza, która przekazuje go do innego wierzchołka, z którego znów protokoły warstwy łącza przekazują go do warstwy sieciowej.
Protokoły warstwy sieciowa - Są one odpowiedzialne za transportowanie pakietów warstwy sieciowej, zwanych datagramami, z jednego hosta do drugiego. Protokoły warstwy transportowej przekazują segment warstwy transportowej i adres docelowy do warstwy sieciowej, niemalże jak nadawanie listu na pocztę z czyimś adresem. Warstwa sieciowa, udostępnia wtedy usługę dostarczenia segmentu do warswy transportowej hosta docelowego. Jednym z protokołów warstwy transportowej jest protokół IP, który definiuje pola w datagramie, tak jak zachowania ich względem. W tej warstwie są też protokoły, które ustalają ścieżki datagramów od źródła do końcowego hosta.
Protokoły warstwy transportowa - Ich rolą jest transportowanie wiadomości wyższej warstwy - application-layer, pomiędzy endpointami. Wyróżniamy dwa takie protokoły - UDP i TCP.
Ramka - jednostka komunikacji w warstwie łącza. Warstwa łącza enkasułuje pakiety z warstwy sieciowej i enkapsułuje je w ramki. Jeśli rozmiar ramki jest zbyt duży, wtedy pakiet może być rozdzielony na mniejsze ramki.
Datagram - pakiety warstwy sieciowej
Segment - pojedynczy fragment danych o pojemności 1024 bajtów
Transport Layer - głównie jak rozumiem tylko dla TCP, tutaj są dane o Flow control oraz rozbijaniu wiadomości na krótsze segmenty
Network Layer - Przesyłanie tych segmentów Transport Layer między hostami (nazywamy to datagramami)
Link Layer - przesyłanie datagramów między punktami w naszej ścieżce (tworzy frame'y)
UDP (user datagram protocol) - protokół bezpołączeniowy, serwer i klient nie tworzą połączenia przed rozpoczęciem transmisji danych; przy użyciu jednego gniazda, możemy komunikować się z wieloma hostami; używamy pary funkcji sendto() i recvfrom(), w których określamy adres hosta do którego wysyłamy lub od którego odbieramy dane; brak gwaracji co do kompletności przesłanych danych
TCP (transmission control protocol) - protokół połączeniowy, serwer i klient tworzą pomiedzy sobą połączenie, gniazdo ma przypisaną informację o parze hostów, które będą się komunikować; odczyty zapisy, wykonywane przy użyciu read, write; odbiór każdego segmentu jest potwierdzany, jeśli brak potwierdzenia - wyślij jeszcze raz
Komunikacja duplexowa (full-duplex) - w każdej chwili możliwe jest przesyłanie danych w obie strony
Pól duplexowa (half-duplex) - w danej chwili komunikacja może odbywać się tylko w jedną stronę (wysyłamy/odbieramy naprzemiennie)
Przetwarzanie równoległe - polega na tworzeniu podprocesów; operuje na innych address space'ach; cieżej jest współdzielić

dane i zasoby programu; jednocześnie przetwarzanie wielu zadań (na osobnych procesach)
Przetwarzanie współbieżne - polega na tworzeniu nowych "unit of execution"w istniejącym już procesie; operuje na tym samym address space; współdzielenie danych nie wymaga żadnych specjalnych operacji; też jednoczesne przetwarzanie zadań, ale na 1 procku, tworzymy po prostu więcej wątków egzekucji procesu, które będą realizowane naprzemiennie
Deadlock - sytuacja, gdzie 2 wątki (lub więcej) czekają na siebie nawzajem aż np. zwolnią zasób, zasygnalizują zakończenie zadania itp.
Livelock - taki deadlock tylko stan procesów się zmienia. 2 wątki, aby uniknąć deadlocka, wyłączają swoje działanie lub podejmująinne akcje, ale robią to w tym samym momencie, więc i tak nikt z nich nie idzie dalej (np. jak 2 zasoby chcą się przepuścić przez drzwi nawzajem)
Starvation - sytuacja, gdzie wątki o większym priorytecie mają dużo do roboty lub jest ich po prostu bardzo dużo i w wyniku tego pojedyncze wątki o mniejszym prio nie dostają czasu procesora
Mount point - katalog znajdujący się w naszym systemie plików, który odwzorowuje innny system plików. Możemy mieć dostęp do plików zamountowanego systemu przez taki katalog
Pseudo-file system - system plików, który tak naprawdę nie ma w sobie plików, tylko daje nam takie przeczucie, że ma. Jądro generuje inormacje, które miałyby się w takim pliku znajdować i serwuje je do aplikacji
Blok - plik dyskowy z ext2 jest podielony na małe sektory zwane blokami. Zazwyczaj mają 1,2,4 kilobajty
Superblock - przechowuje metadane o systemie plików. Główna kopia jest na offsecie 1024
Grupa bloków - bloki są złączone w grupę by dostęp był szybszy i powodował mniej fragmentacji. Info o każdej grupie jest w tabelach deskryptorów zaraz za superblokiem
Tabela deskryptorów grup bloków - opisuje zawartość i metadane wszytskich grup bloków. Ich kopia jest w parze z kopią superbloków
Hard link - pointer na inode, jedynie może zmienić się nazwa pliku w katalogu
Symbolic link - specjalny plik który wskazuje na inny plik. Zawiera absolutną ścieżkę do pliku
Przydatne komendy, które używaliśmy na przestrzeni SO
strace - Śledzi wywołania systemowe oraz sygnały w procesie
waitpid - czekaj aż zginie dziecko o danym PID
proc - my używaliśmy proc/PID, pozwalało na modyfikację/odczyt parametrów procesu
lsof - pokaż wszystkie otwarte pliki w systemie plików
fork - robimy widelec pog, no chyba standard
dup i dup2 - stwórz kopię deskryptora plików
lseek - zmiana wskaźnika skąd czytamy plik (jakieś fiku miku można robić że np cofać na początek)
execve - zmieniamy "mózg"procesu,
sigwait - czekaj aż przyjdzie dany sygnał
sigprocmask - zrób coś z maską sygnałów
ioctl - systemcall do zarządzania urządzeniami wejścia/wyjścia
stat - i-node, kto ostatnio modyfikował itp itd, (fstat to jak podamy mu fd)
sbrk - zwiększ HEAP (STERTĘ) procesu
stos - jest na górze gdy **sterta** jest na dole.