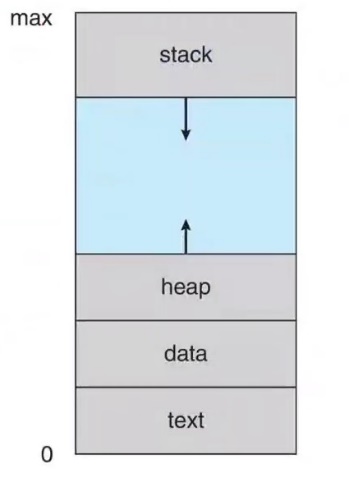
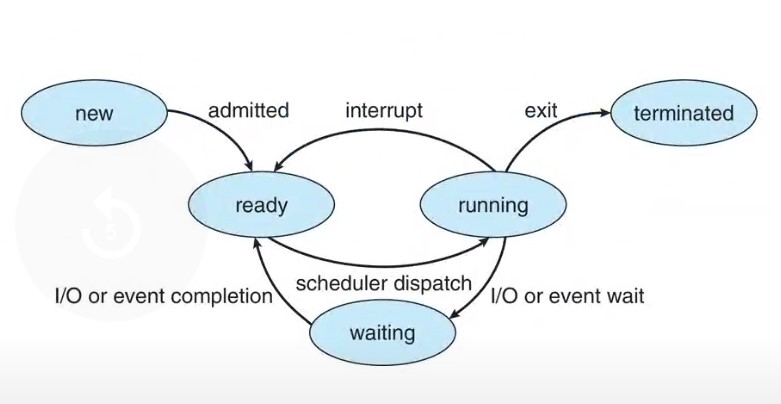
**Menaxhimi i proceseve**

Elementet që e përbëjnë një *proces* janë : *Kodi i programit/seksioni i tekstit*, *Progam counter* – Tregues se në cilën pjesë procesi është duke u ekzekutuar, *Stack* – Hapsirë e përkohshme në të cilën vendosen të dhëna si parametrat, variablat lokale etj, *Heap* – Hapsirë e memories e alokuar gjatë run-time, *Seksioni i data* – Hapsir kur vendosen variablat globale.

*Madhesia e stack dhe heap mund të ndryshojë ndërsa data dhe text janë statike.*

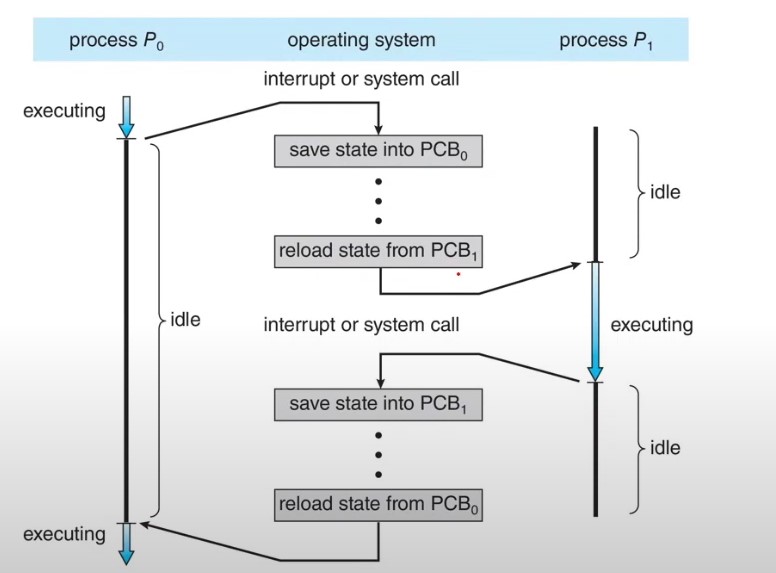
Një proces kalon në disa *gjendje/state* : 1.New – Këtë gjendje e ka në momentin kur krijohet. 2.Running - Këtë gjendje e ka në momentin kur fillon ekzekutimin. 3.Waiting - Këtë gjendje e ka nëse është duke pritur për diqka të ndodhë. 4.Ready - Këtë gjendje e kur është duke pritur që CPU të i jep vëmendje ta ekzekutoj. 5.Terminated - Këtë gjendje e ka në momentin kur ekzekuton komandën e fundit të kodit pra mbraon punën. 

Kur një proces krijohet OS i krijon një tabel të quajtur *Process Control Block (PCB),* PCB është e nevojshme shkaku që sic pamë gjendja e procesit mund të ndryshojë dhe kur ndryshon ne duhet patjetër ta mbajmë në mend gjendjen e tij. Në PCB mund të gjejmë informata si : *Program counter*, *informata për CPU scheduling* (kjo informata varion vartësisht nga alogritmi që përdorim për cpu scheduling, mund të jetë informat për prioritetin, pointer në ready queue etj), *Informata për menagjimin e memories* (memoria e alokuar për atë process), *Loggin/Accounting* (Informata si përdorimi i CPU, clock të përdorur, koha nga fillimi deri tash, time limiti), *Informata rreth I/O* (Paisjet hyrse dalse të alokuara për atë process).

Proceset me shumë threda kanë më shumë Program counter (për secilin thread).

*Context Switching*  - ndodhë kur CPU kalon nga ekzekutimi i një procesi në një ekzekutimin e një procesi tjetër por duke e ruajtur komplet gjendjën e procesit që po largon dhe duke e kthyr në gjendjën e njëjtë kur ndodhë context switching tjetër.

Rrjedha e punës kur ndodhë *Context Switching* : 1.CPU është duke e ekzekutuar një proces P. 2.Papritur ndodhë ndonjë interrupt ose ndonjë System Call. 3.OS e ruan PCB e P për ta mbajtur gjendjen e tij. 4.Bëhet reload PCB i procesit Q i cili po vjen në ekzekutim.



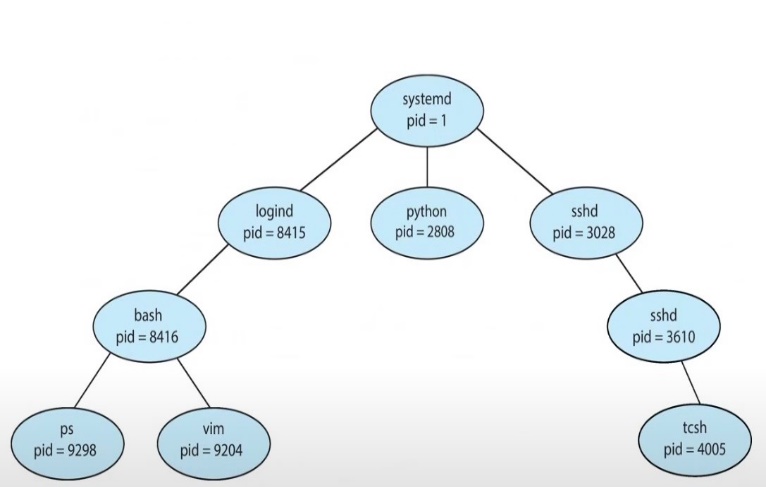
Në mobile proceset ndahen në *Foreground* procese të cilat janë duke u ekzekutuar dhe useri është duke i përdorur dhe në *Background* procese të cilat janë duke u ekzekutuar në background mirpo numri i gjërave që mund të bëjnë janë të limituara.

Operacionet kryesore që OS duhet ti ofroj për proceset janë : Krijimi dhe terminimi i proceseve.

*Krijimi i procesit* – Kur një proces krijohet i gjenerohet një identifikues unik (PID), procesi i parë i krijuar quhet proces prind, proceset që krijohet nga ai proces quhet procese fëmijë.

Vartësisht nga OS varion mënyra e ndarjes së resuereve në mes prindit dhe fëmiut : 1.I ndajnë të gjitha resurset. 2.I ndajnë disa resurse. 3.Nuk ndajnë resurse fare.

I njëjti varjim vlen edhe sa i përket mënyrës së ekzekutimit : 1.Procesi prind dhe ai fëmij ekzekutohen në mënyrë konkurrente. 2.Procesi prind pret ekzekutimin e procesit fëmijë. (Këtë aproach e përdorin UNIX dhe Linux).

Në linux procesi i parë i krijuar është systemd dhe të gjitha prcoeset tjera janë fëmijë të systemd, ky procesi i bën wait() të gjith fëmijët e tij, kur terminohen proceset fëmijë terminohet edhe systemd.

*Termini i procesit* : Një proces terminohet kur ekzekuton instruksionet e fundit dhe pastaj kërkon nga OS me anë të system calls exit() ta fshijë atë, në momentin e ekzekutimit të exit() statusi i ati procesi i kthehet prcoesit prind me anë të system call wait(). Procesi pos tjerash mund të terminohet edhe nga prindi i tij kur ai i thërret system call abort(), një gjë e till mund të ndodhë për arsye se procesi fëmij konsumon më shumë resurse se që ka të alokuara ose nëse thjesht ai proces nuk nevojitet më, ose ose nëse terminohet procesi prind atëher OS nuk lejon ekzistimin e procesve fëmijë.

*Cascade Terminimi* - Ndodhë shkaku i asaj që OS nuk lejon ekzistimin e proceseve fëmij pa proceset prind.

Për me pritë ekzekutimin e proceseve fëmij Procesi prind përodr sys callin wait(). Nëse asnjë prind nuk është duke wait() një proces fëmij ai proces quhet *zombie*. Nëse terminohet procesi prind i cili ka thirrur wait() në një prcoes para se ai proces të kthej statusin në wait, ai proces quhet *jetim*.

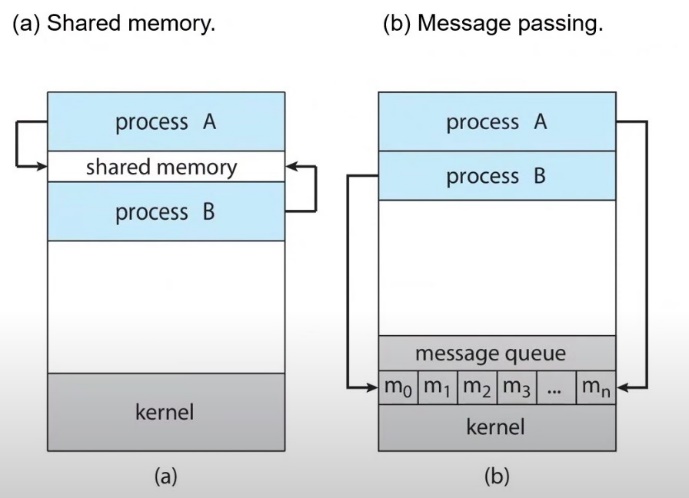
Në andoroid proceset listohen në këtë rënditje në bazë të rëndësis së tyre : 1.Foreground Proceset. 2.Visible Proceset. 3.Service Proceset. 4.Background Proceset. 5.Empty Proceset.

Google chrome i klasifikon proceset e veta në 3 lloje : 1.Browser proceset – mirren me UI, disk, network... 2.Render proceset – mirren me HTML, JS,pra me renderimin e web. 3.Plugin proceset – përsecilin plugin si Adobe Flash krijohet nga një proces që mirret me të.

Në OS proceset mund të jenë *Indepedent* *Proceset*(Për ekzekutimin e tyre nuk kanë nevoj për proceset e tjera) dhe *Cooperating Proceset* (Gjatë ekzekutimit të tyre duhet patjetër të kenë ndërveprim me proceset e tjera).

Arsyet pse përdoren *Cooperating Proceset* janë : *Ndaraj e informatave, Shpejtësia e kompjutimit, Modulariteti dhe Komoditeti*.

*Cooperating proceset* kanë nevoj për komunimik kanë nevoj për *Inter prcoess communication* (IPC). Ekzistojnë dy modele të IPC : 1.*Shared Memory* (Kontrollohet nga useri). 2.*Message Passing* (Kontrollohet nga OS).



*Cooperatin proceset*  në një paradigmë ballafaqohen me problemin prodhues-konsomatorë. Ku procesi prodhues prodhon infomrata të cilat konsumohen nga procesi konsumatorë. Për zgjedhien e kti problemi kemi dy mundësi : 1.*Unbouded-Buffer* – Bufferi ka madhësi të pa kufizuar e cila mundëson që prodhuesi mos të pres asjnëherë gjithmon është duke prodhuar, ndërsa konsumatori pret në raste se baferi është i zbrazët. 2.*Bounded-Buffer* – Buferi ka madhësi fikse e cila bën që prodhuesi të pres në rast se bufferi është full dhe konsumatori të pres në rastë se bufferi është i zbrazët.

Problemet me *Shared-memory* : Ofrimimi i mekanizmave për të lejuar userin sinkronizimin e qasjes së proceseve në memorien e përbashkët dhe ambienti i kufiziuar ku mund të veprojë ( Në një kompjuter është i mirë, por në kompjuter të shpërndarë nuk është i përshtatshëm).

Nëse *Cooperatin proceset* përdorin IPC – *Message passing* atëher OS duhet të ofrojë dy opracione kryesore : send(message) dhe recive(message).

Për të filluar komunikimin dy cooperatin procesee me message passing duhet të ndjekin këta hapa : 1.Të krijojnë një lidhje/link komunikuese. 2.Të dërgojnë dhe pranojnë mesaszhe.

Nëse implementojmë message passing ne do të prballemi me probleme si : *Qfarë lloji të lidhjes të krijojmë, sa lidhje mund të vendosen midis dy proceseve, cfarë kapaciteti, a është lidhja/linku një drejtimësh apo shumë drejtimësh*.

Implementimi i Lidhjeve/Links komunikues ndaheht në pjesën : 1.Fizike : Shared memory, Hardware Bus, Network. 2.Logjike : Direkt ose indirekt, Sinkronizuara ose pa sinkronizuar, Buffer automatik ose ekspilicit.

*Komunikimi Direkt* : Nëse procesi dërgon mesazh në send(P, msg) system call përveq msg duhet të oforjë si parametër edhe procesi tek i cili po e dërgon at msg, po ashtu edhe procesi i cili e pranon msg duhet që në recive(Q, msg) përveq msg që e pranon duhet të ofrojë edhe procesin nga i cili po e pranon msg.  *Vetitë e komunikimit direkt* : 1.Lidhja/Link krijohet automatikisht. 2.Dy procese mund të kenë vetëm një Lidhje/Link. 3.Një lidhje/link lidh vetëm dy procese. 4.Link mundet me qenë undirect ose bidirect.

*Komunikimi Indirekt* : Mesazhet dërgohen me anë të *Mailbox(Port)*. Secili *mailbox* ka Id unike dhe nëse procest duan të komunikojnë duhet të kenë *mailbox* të përbashkët. *Vetitë e komunikimit indirekt* : 1.Lidhja krijohet vetëm nëse ka mailbox të përbashkët. 2.Një lidhje mund të lidh shum procese. 3.Midis dy proceseve mund të ketë shumë lidhje. 4.Lidhja mund të jet undirect ose bidirect.

Nëse 3 procese ndajnë të njëjtin *Mailbox* , pra kemi procesin P1,P2,P3. Procesi P1 dëron një meazh në mailbox. Këtu shfaqet problem nuk e dim se cili proces do ta pranoj mesazhin P2 apo P3. Për ta zgjedhur problemin e mailbox ne kemi 3 mundësi : 1.Kufizojmë numrin e proceseve që mund të lidh një Lidhje/Link në dy procese. 2.Lejojmë që vetëm një proces mund ta ekzekutoj recive() në një kohë. 3.Lejojmë që OS të zgjedh se cili nga proceset ta pranoj mesazhin e pastaj ta njofoj dërguesin se cili e ka pranuar.

Nëse përdorim komunikimin indirekt OS duhet të mundësoj proceseve kryerjen e këtyre operacioneve : 1.Krijimi i mailbox. 2.Dërgimi/pranimi nga mailbox. 3.Fshirja e mailbox.

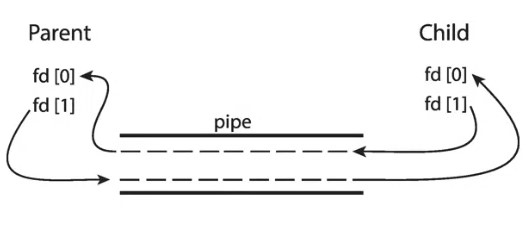
*Komunikimi sinkron* (Blocking) – Procesi dërgues bllokohet derisa procesi pranues të ket pranuar mesazhin. Procesi pranues bllokohet për sa koh që nuk ka mesazhe për të pranuar.

*Komunikimi asinkron* (No Blocking) – Procesi dërgues dërgon mesazhin dhe vazhdon. Procesi pranues pranon ose mesazh valid ose pranon null.

Zakonisht forma e komunikimit sinkron kombinohet me atë a sinkron.

*Bufferi* është hapsirë ku vendosen të dhënat që janë duke kaluar nga një vend në një vend tjetër, në rastin e pasimit të mesazheve luan rolin e një queue që mbushet vazhdimisht me mesazhe. Janë 3 lloje të bufferave që përodren : *1.Zerro Kapacitet –* Buferi nuk ka hapsirë dmth Dërguesi duhet me prit deri sa mesazhi të pranohet. *2.Bounded Capacity* – Buferi ka madhësi të kufizuar dërguesi pret në rastë se bufferi është full. *3.Unbouded Capacity* – Buferi ka madhësi të pa kufizuar dërguesi kurr nuk pret.

*Pipes* janë kanale ose gype të cilat përdoren nga dy procese për të komunikuar. Ekzistojnë dy lloje të *Pipes* : 1.Ordinary Pipes : Në windows njihet si *annonymous pipes*, janë një drejtimëshe të dhënat shkruhen në njërën anë dhe pranohen në anën tjetër, proceset për të përdorur *ordinary pipes* duhet të kenë relacion *prind-fëmijë,* vetëm dy procese mund të komunikojn me një *ordinary pipe*



2.*Named Pipes* : Janë dy drejtimëshe, proceset nuk janë të obliguara të kenë relacion *prind-fëmijë*, me një *named pipe* mund të komunikojnë shumë procese. Kto pipes gjejnë zbatim në windows dhe unix.

Në ambientet klient-server komunikimi i proceseve realizohet në dy forma : *Sockets* ose *Remote procedure calls*.

*Sockets* janë end-point i komunikimit në mes të pasijeve në rrjetë. Një socket formohet nëse bashkojmë një IP adressë me një Port. përshembull 161.25.19.8::1625

Portat janë hyrje në hostin me një Ip të caktuar.

Portat më të vogël se 1024 janë të rezervuar.

