FACULTATEA DE ELECTRONICĂ,TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI, UPB 2012

proiect:Singularity OS

realizat de: Ciobănică Radu-Constantin,
Grupa 431A

Cuprins

1. Introducere

Singularity este un proiect de cercetare al grupului de cercetare Microsoft care-a fost proiectat de la zero cu scopul principal al dependabilitatii, dar si al imbunatatirii stabilitatii si producerea unei platforme software robusta.

Odata cu rata exponentiala a progresului, evolutia hardware atrage de la sine schimbari fundamentale in sisteme si aplicatii. Software-ul evolueaza intr-un mod invers hardware-ului, astfel produce rareori oportunitati pentru imbunatatiri fundamentale. Evolutia software aduce totodata prilejul regandirii si integrarii unor idei vechi sau nefolosite.

Sistemul de operare Singularity este in continuare in dezvoltare, si este privit ca baza pentru un sistem mai stabil si aplicatii software pe masura.

Un aspect cheie al acestui sistem de operare este modelul de extensie bazat pe procesele izolate la nivel software (SIPs – "*Software-isolated processes"*), care incapsuleaza parti ale unei aplicatii sau a sistemului si furnizeaza mascari ale informatiei, izolatii ale esecurilor/erorilor, si interfete puternice. SIP-urile sunt folosite in tot sistemul de operare si in aplicatiile software. Acestea sunt procesele ce tin de sistemul de operare in cazul Singularity. Tot codul din exteriorul nucleului se executa intr-un SIP. Astfel, SIP-urile difera de procesele sistemelor de operare conventionale in mai multe directii:

- SIP-urile sunt spatii de obiecte inchise si nu spatii de adrese. Doua procese de tip Singularity nu pot accesa simultan un obiect.
- Un proces nu poate incarca dinamic sau genera cod.
- SIP-urile nu se bazeaza pe hardware-ul de management al memoriei pentru izolare. Mai multe SIP-uri se pot afla intr-un spatiu de adrese fizic sau virtual.
- Comunicatiile intre SIP-uri se fac prin canale de ordin inalt, bidirectionale. Un canal specifica protocolul de comunicatii si valorile transferate, dupa care ambele sunt verificate.
- SIP-urile nu sunt scumpe din punct de vedere al costului de productie.
- SIP-urile sunt create si finalizate de sistemul de operare, astfel incat, in momentul finalizarii, resursele SIP-ului pot fi valorificate eficient.
- SIP-urile sunt executate independent, chiar si la nivelul amplasamentului de date, timpului de executie si a colectorilor de gunoi(date reziduale).

Nucleul sistemului de operare Singularity consta aproape in totalitate in cod sigur, iar restul sistemului, executat in SIP-uri, este scris in cod sigur verificabil, incluzand toate driver-ele dispozitivelor, procesele de sistem si aplicatii. In timp ce tot codul ce nu face parte din *aria de incredere* trebuie sa fie verificat si categorisit ca *sigur*, parti ale nucleului si a timpului de executie, inglobate in *baza de incredere*, nu sunt sigure. Siguranta limbajului protejeaza *baza de incredere* de cod ce nu se afla in *aria de incredere*.

Printre contributiile cheie ale sistemului de operare Singularity amintim:

- Constructia unui sistem si a unui model de aplicatie numit proces izolat software, care utilizeaza cod sigur verificat pentru a implementa o legatura puternica intre procese fara mecanisme hardware. Cum SIP-urile au costuri de productie mici, sistemul si aplicatiile pot suporta legaturi de izolatie mai bune si un model de izolatie mai puternic.
- Un model consistent de expansiune pentru sistem si aplicatii care simplifica modelul de securitate, imbunatateste dependabilitatea si recuperarea in caz de esec, creste optimizarea codului, si creste eficienta instrumentelor de programare si testare.
- Un mecanism rapid si verificabil de comunicatie intre procesele unui sistem, care pastreaza izolarea si independenta proceselor, si in acelasi timp permite proceselor sa comunice corect si la un cost scazut.
- Suport de limbaj si compilator pentru a construi un intreg sistem in cod sigur si sa verifice comunicatiile intre procese cu management explicit de resurse.
- Eliminarea distinctiei dintre un sistem de executie al sistemului de operare si un sistem de executie scris intr-un limbaj sigur, precum JVM(Java) si CLR(Microsoft).

2. Arhitectura sistemului Singularity

In figura 1 se pot distinge cele trei componente cheie ale sistemului de operare Singularity: un nucleu(kernel), procesele izolate software, si canalele. Kernel-ul furnizeaza functionalitatea de baza a sistemului, incluzand management-ul memoriei, crearea si terminarea proceselor, operatiile canalelor, programarea, si I/O. Precum alte micronuclee, mare partea a functionalitatii sistemului si a extensibilitatii exista in procesele din afara nucleului.

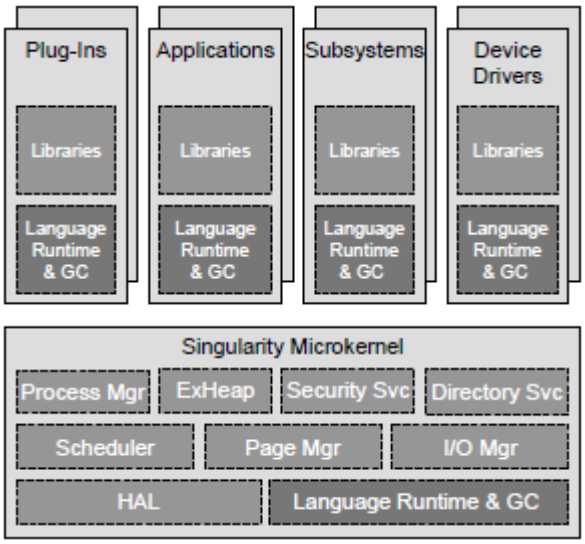

figura 1. Arhitectura Singularity[1]

2.1 Baza de incredere

Codul, in cadrul sistemului Singularity este fie verificat(verified), fie de incredere(trusted).

Siguranta tipului si memoriei codului verificat este evaluata de compilator. Codul care nu poate fi verificat trebuie sa fie acceptat de sistem si este limitat pana la nivelului de abstractizare hardware(HAL), nucleu, si parti ale sistemului de executie. Majoritatea codului continut de nucleu este sigur, dar portiuni sunt scrise in assembler, C++, si in C#(limbaj nesigur).

Restul codului este scris intr-un limbaj sigur, tradus(nesigur->sigur) prin intermediul Microsoft Intermediat Language(MSIL), dupa care este compilat pentru x86 cu ajutorul compilatorului Bartok. In ceea ce priveste dezvoltarea acestei sectiuni, se urmareste ca pe viitor sa se foloseasca un limbaj assemble care sa verifice direct rezultatele compilatorului, astfel reducand numarul de etape de executie ale procesului.

2.2. Nucleul

Nucleul sistemului Singularity este o componenta privilegiata care controleaza accesul la resursele hardware, aloca si recupereaza memoria, creeaza si programeaza firele de executie, furnizeaza firele de sincronizare din interiorul proceselor, si gestioneaza I/O. Este scris intr-un amestec de cod sigur si nesigur(C#) si ruleaza in propriul spatiu de obiecte prevazut cu colector de gunoi.

Fata de mecanismul canalelor de transmitere a mesajelor, acesta are un plus, procesele comunica cu nucleul printr-o *interfata de aplicatie binara*(ABI) care invoca metode statice in codul nucleului. Aceasta interfata urmeaza modelul restului sistemului, izoland nucleul si procesand spatiul obiectelor. Toti parametrii ABI-ului sunt valori, nu pointeri, astfel ca nucleul si colectorii de gunoi ai proceselor n-au nevoie de coordonare. Singura exceptie este in locatia metodelor ABI. Colectorii de gunoi nu relocheaza cod.

ABI mentine starea de izolare constanta in intregul sistem: un proces nu poate schimba starea unui alt proces folosind ABI. Cu numai doua exceptii, apelarea ABI poate influenta numai starea procesului in care este implicat. Cele doua exceptii modifica starea unui proces derivat („child process” al unui „parent process”) inainte sau dupa, dar nu in timpul executarii. Prima exceptie consta in crearea unui proces derivat, care specifica codul incarcat pentru derivat inainte de inceperea executarii. Cea de-a doua exceptie este aceea de a opri un proces derivat, care isi revendica resursele dupa ce executarea a incetat. Izolarea starii asigura ca un proces Singularity are control unic asupra starii sale.

Nucleul exporta obiecte de sincronizare- mutex-uri(excludere mutuala – se pune problema ca doua procese se afla in sectiunea critica in acelasi timp),

evenimente cu resetare auto si manuala – pentru a coordona firele de executie dintr-un proces.

Un fir manipuleaza aceste constructii printr-un „handle” folosind un „hard typing” opac ce indica spre *handle table*-ul nucleului.

„Strong typing” si „weak typing” sunt termeni folositi in IT pentru a descrie modul in care restrictiile privind modul in care operatiunile care implica valori de diferite tipuri de date pot fi amestecate.

„Strong typing”–ul previne un proces in a schimba sau corupe „handle”-urile. In plus, sloturile din „handle table”-uri sunt recuperate doar cand un proces se termina, pentru a preveni procesul in a elibera un mutex, retinandu-i „handle”-ul, si folosindu-l sa manipuleze obiectul altui proces. Singularity refoloseste intrarile de „table”dintr-un proces. In acest caz, retinerea unui „handle” poate cauza o eroare „canceroasa” intr-un proces.[3]

2.2.1 Versionarea ABI [4]

ABI-ul nucleului contine mai multe versiuni. Pentru a identifica intr-un mod explicit informatiile despre versiunea ABI in fiecare program, Singularity ofera o cale clara pentru evolutia sistemului si compatibilitatea inversa.

Codul din cadrul unui proces este compilat si in acelasi timp comparat cu o interfeta compilata ABI intr-un spatiu de nume care specifica in mod explicit versiunea. De exemplu, *Microsoft.Singularity.V1.Threads* este spatiul de nume care contine functionalitatea legata de fir pentru versiunea primara a ABI-ului. Codului sursa al procesului specifica spatiul de nume ce contine versiunea dorita de ABI. Codul binar al procesului contine explicit referinte metadata la versiunea specifica a ABI-ului.

In ceea ce priveste instalarea, un program este instalat doar daca versiunea de ABI este suportata. Daca versiunea este acceptata,ansamblul interfetei ABI este inlocuit de ansamblul implementarii, care furnizeaza o imlementare pe partea de procese a versiunii ABI-ului, corespunzatoare nucleului. Un „nou”(relativ nou, avand in vedere ca nu s-a mai scris cod din mai, 2010) sistem Singularity poate popula spatiu de nume dintr-o versiune anterioara cu o librarie de functii de compatibilitate. Alternativ, codul compatibil poate rula in nucleu, deoarece nucleul poate suporta cu usurinta implementari ABI multiple in spatii de nume distincte.

Prima versiune a ABI-ului nucleului contine 126 intrari.

2.2.2 Planificator [5]

Singularity suporta un planificator inlocuibil din punct de vedere al timpului de compilare. Pana la momentul de fata au fost implementati patru

planificatori: Rialto, un planificator bazat pe o laxitate cu resurse multiple, un planificator de tip „round-robin”(round-robin = un aranjament de selectie a elementelor dintr-un grup intr-o ordine rationala, de exemplu: din varful listei pana la baza, sau invers) si un planificator de tip „round-robin” cu o latenta minima.

Planificatorul Rialto vine cu urmatoarele caracteristici:

- ❖ Planificare in timp real, cu o latenta scazuta;
- ❖ Constrangerile de timp dinamice;
- ❖ Prioritizarea CPU-ului in functie de activitate;
- ❖ Administrarea resurselor in timp real, intr-un mod extensibil;

Planificatorul round-robin cu latenta minima este optimizat pentru un numar mare de fire ce comunica frecvent. Planificatorul mentine doua liste de fire in executie. Prima se numeste lista „deblocata”, si contine fire ce au devenit executabile de curand. A doua se numeste lista “preempted”, care contine fire executabile care au fost pre-inlocuite. Cand se face selectia firului ce urmeaza a fi rulat, planificatorul sterge firele din lista “deblocata” in ordinea FIFO. Dupa ce lista “deblocata” este goala, planificatorul sterge urmatorul fir din lista “preempted”. Oricand intervine o intrerupere in timer-ul planificatorului , toate firele din lista “blocata”sunt mutate la sfarsitul listei “preempted”, urmate de firul care rula in momentul declansarii timer-ului. Primul fir din lista “deblocata” este programat si timer-ul planificatorului este resetat.

Efectul de plasa a politicii de planificare a celor doua liste favorizeaza firele trezite de un mesaj, care fac o cantitate mica de munca, apoi trimit unul sau mai multe mesaje altor procese, dupa care inceteaza orice, pana la receptionarea unui mesaj. Acesta este un comportament comun pentru firele care se folosesc de manevrarea in bucla a mesajelor. Pentru a evita resetarea costisitoare a timer-ului planificatorului, firele din lista “deblocata” mostenesc cuantumul programat al firului ce le-a deblocat. Combinat cu politica celor doua liste, mostenirea cuantumului este eficienta in particular pentru ca Singularity poate sari de la un fir la altul in aproape 394 de cicluri.

2.3. Procese

Sistemul Singularity ruleaza intr-un singur spatiu virtual de adrese. Hardware-ul ce tine de memoria virtuala este folosit pentru a proteja paginile, de exemplu prin maparea primilor 16K al spatiului de adrese sa capteze referinte de pointeri nuli. In interiorul sistemului Singularity, spatiul adreselor este partitionat logic intr-un spatiu obiect al nucleului, un spatiu obiect pentru fiecare proces, si schimbul Heap pentru datele canalelor. O decizie de design ominprezenta este invarianta memoriei independente: pointerii spatiului dintre obiecte indica doar in schimbul Heap.

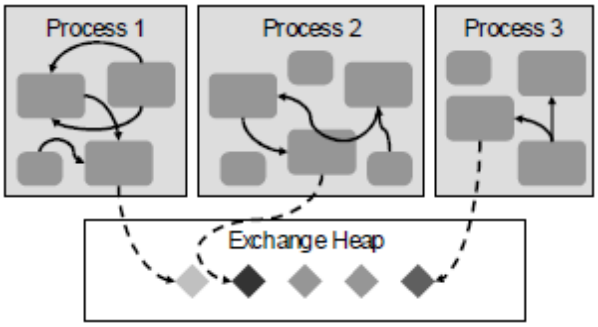

fig.2: Schimbul Heap[2]

In particular, nucleul nu are pointeri in spatiul obiectelor proceselor, si nici procesele nu au pointeri catre obiectele altui proces. Aceasta invarianta asigura ca „gunoiul" fiecarui proces poate fi colectat si terminarea proceselor se face fara cooperarea altor procese.

Nucleul creeaza un proces alocandu-i suficienta memorie pentru a incarca si executa imaginea dintr-un fisier stocat in formatul PE(Microsoft's Portable Executable). Singularity efectueaza apoi realocari si debug-uri, incluzand functiile ABI-ului nucleului. Nucleul incepe noul proces prin crearea unui fir ce ruleaza la punctul de intrare al imaginii, al carui cod este categorisit „trusted" si care apeleaza stiva si managerul paginilor pentru a initializa procesul.

Un proces obtine spatiu de adrese aditional prin apelarea managerului de pagini, care returneaza pagini noi, nepartajate. Aceste pagini nu trebuie sa fie adiacente la spatiul adreselor procesului existent, deoarece colectorii de gunoi nu au nevoie ca spatiul de adrese sa fie contiguu, desi ele ar putea nevoie de regiuni contiguu pentru obiecte mari sau vectori. Aditional memoriei, care contine codul procesului si data heap, procesul are o stiva per fir si poate accesa schimbul Heap.

2.4 Colectarea gunoiului

Colectarea gunoiului reprezinta o componenta esentiala a celor mai multe limbaje sigure, intrucat previne erorile de de alocare a memoriei. In Singurality, colectorul de gunoi se aplica atat nucleului cat si pentru spatiile obiectelor proceselor.

Numarul mare de algoritmi de colectare a gunoiului si experienta sugereaza ca nici un alt colector de gunoi nu este adecvat pentru tot sistemul sau pentru tot codul de aplicatie. Arhitectura sistemului de operare Singularity decupleaza algoritmul, structurile de date si executia colectorului de gunoi pentru fiecare proces, astfel incat poate fi selectat sa acomodeze comportamentul codului in proces si sa lucreze fara a necesita coordonare globala. Cele patru aspecte ale Singularity care fac acest lucru posibil sunt : fiecare proces este un mediu inchis cu propriul sau suport de timp de lucru ; pointerii nu intersecteaza limitele procesului sau ale nucleului, asa incat

colectorii nu trebuie sa aiba in vedere acele indicatoare de intersectare a spatiului ; mesajele sau canalele nu sunt obiecte, rezultand ca acordul privind configuratia memoriei este necesar numai pentru mesaje si alte date in schimbul Heap ; si nucleul controleaza alocarea paginii de memorie, care furnizeaza o relatie pentru coordonarea alocarii de resurse.

Sistemele ce se ocupa cu timpul de lucru al lui Singularity suporta cinci tipuri de colectori – semi-spatiul generativ, compactarea glisant-generativa, o combinatie adaptativa intre cele doua mai sus mentionate, curatator de amprente si curatator de amprente concomitent. In momentul de fata, Singularity il foloseste pe cel din urma pentru codul sistemului, deoarece are foarte scurte momente de pauza in timpul colectarii. Cu acest colector, fiecare fir are o lista de izolare liber, care elimina in mod normal sincronizarea firelor. O colectare a gunoiului este declansata la un prag de alocare si executa intr-o colectie independenta de fire, actiune care marcheaza obiectele accesibile. In timpul unei colectari, colectorul opreste fiecare fir pentru a-i scana stiva, acesta introducand un timp de pauza de mai putin de 100 microsecunde pentru un stoc tipic. Suprasarcina acestui colector este mai mare decat a colectorilor non-concomitent, astfel ca in aplicatii se foloseste un colector non-concomitent de stergere a amprentelor mai simplu.

Fiecare SIP are propriul sau colector care este responsabil exclusiv pentru colectarea obiectelor din spatiul obiectelor. Din perspectiva colectorului de gunoi, cand un fir de control intra sau paraseste o aplicatie (sau nucleul) este tratat similar cu un apel catre sau dinspre codul nativ in mediul conventional al gunoiului colectat. Colectarea gunoiului pentru diferite spatii de obiecte poate fi, astfel programata si rulata complet independent. Daca o aplicatie utilizeaza un colector de tipul celor care opresc aplicatia, un fir este considerat oprit cu privire la spatiul obiectual al aplicatiei, chiar daca acesta lucreaza in spatiul obiectual al nucleului datorita unui apel al nucleului. Firul de executie, in orice caz, este oprit din a se intoarce in spatiul de procese al aplicatiei pe durata colectarii.

2.4.1 Gestionarea stivei

Intr-un mediu in care gunoiul este colectat, stiva unui subproces contine referinte la obiecte care sunt potentiale radacini(root) pentru un colector. Apelurile in nucleu sunt executate pe stiva subprocesului unui utilizator si poate depozita pointeri catre nucleu in aceasta stiva. La prima vedere, aceasta pare sa violeze memoria invariabila si independenta prin crearea unor pointeri ce se regasesc dealungul procesului, si, cel putin, incurca colectatarile gunoiului executate pentru utilizator si nucleu.

Pentru a evita aceste probleme, Singularity delimiteaza granitele dintre cadrele fiecarui spatiu de stiva, asadar un colector de gunoi nu are nevoie de referinte privind celalalt spatiu. La un apel de tip incrucisat (*proces-> nucleu* sau *nucleu-> proces*), Singularity salveaza registrele apelurilor salvate ale

emitentilor intr-o structura speciala in stiva, care de asemenea demarcheaza un apel de tip incrucisat. Aceste structuri marcheaza granita regiunilor stivelor care apartin fiecarui spatiu obiectual. Fiindca apelurile din nucleu ABI nu paseaza pointeri la obiecte, un colector de gunoi poate omite cadre apartinand celuilalt spatiu.

Acesti delimitatori faciliteaza de asemenea terminarea curata a proceselor. Cand un proces este finalizat, subprocesele sale sunt oprite si nucleul le intinde fiecaruia cate o exceptie, care omite si dealoca cadrele de stoc ale procesului.

2.5 Canale

Procesele Singularity comunica exclusiv prin trimiterea mesajelor prin canale, acestea reprezentand o conexiune bidirectionala si de tip comportamental intre doua procese. Mesajele sunt colectari de valori sau blocuri de mesage in Schimbul Heap atasate care sunt transferate de la un proces de trimitere la unul de primire. Un canal este stabilit de un contract care specifica formatul mesajului si secventele mesajelor valide de-a lungul canalului.

Un proces creeaza un canal prin invocarea unei metode statice NewChannel a unui contract, care readuce cele doua capete ale canalului-asimetric atribuite ca exportator si importator – in parametrii de output :

C1.Exp importCh ;
C1.Imp exportCh ;
C1.NewChannel (out importCh, out exportCh);

Procesul poate pasa unul sau ambele capete la alte procese pe canalele existente. Procesul care primeste un capat are un canal la procesul care detine celalalt capat corespunzator. De exemplu, daca un proces de aplicatie vrea sa comunice cu un sistem, aplicatia creeaza doua capete si trimite o cerere continand un capat la serverul sistemului, care trimite mai departe capatul la serviciu, astfel stabilind un canal intre proces si serviciu.

Transmisiunea pe un canal este asincrona. O primire blocheaza in mod sincron pana cand ajunge un mesaj specific. Folosind caracteristici lingvistici, un subproces poate astepta primul dintr-un set de mesaje pe un canal sau poate astepta seturi specifice de mesaje de la diferite canale. Cand datele sunt transmise printr-un canal, detinerea trece de la procesul de trimitere, care poate sa nu mentina o referinta a mesajului, la procesul de primire. Invariabla detinerii este intarita de limbaj si de sistemele de run-time si serveste pentru 3 cauze.

Prima este pentru a preveni partajarea intre procese. Cea de-a doua este sa faciliteze analiza programelor statice prin eliminarea pointerilor eronati ai mesajelor. Cea din urma este de a permite flexibilitatea implementarii prin asigurarea unei semantici a transmiterii mesajelor ce poate fi implementata prin copierea sau pasarea pointerilor.

2.5.1 Implementarea canalelor

Capetele canalelor si valorile transferate de-a lungul canalelor apartin Exchange Heap. Capetele nu pot apartine spatiului obiectual al procesului, fiindca sunt transmise cu ajutorul canalelor. In mod similar, data transmisa printr-un canal nu poate apartine unui spatiu obiectual deoarece ar viola memoria invariabila si independenta. Cel ce trimite mesajul transmite posesia prin stocarea pointerului mesajului la capatul celui ce primeste, la o locatie determinata de starea curenta a protocolului de schimb al mesajului. Aceasta abordare permite in mod natural o implementare «zero copy » a unui stoc I/O. De exemplu, pachetele de internet pot fi transferate prin canale multiple, printr-un stoc de protocol si intr-un proces de aplicatie, fara copiere.

2.5.2 Sisteme de executie personalizate

Arhitectura Singularity permite SIP-urilor sa gazduiasca diverse sisteme de executie, care permite unei executii sa fie personalizata pentru fiecare proces. De exemplu, SIP-urile care ruleaza cod secvential nu au nevoie de suport pentru sincronizarea firelor de executie, suport necesar SIP-urilor cu fire multiple. SIP-urile fara obiecte ce necesita finalizare(sau ce prezinta finalizatori care nu acceseaza data partajata intre fire) s-ar putea sa nu aiba nevoie de un finalizator separat pentru a observa semantica limbajului ncesara pentru finalizatori. SIP-urile cu o anumita strategie de alocare pot fi capabile sa pre-aloce sau sa aloce intr-un model stiva memorie pentru toate obiectele utilizate, evitand nevoia unui colector de gunoi in executia SIP-ului.

3. Concluzii

Programarea in cod sigur ofera o multitudine de avantaje in realizarea unui software sigur care e imun la „gaurile” de nivel mic ale securitatii care sunt specifice codurilor C si C++. Datorita acestelor beneficii practice, limbajele sigure devin din ce in ce mai populare. Sistemele de operare conventionale nu ofera suport special pentru programele sigure, si nici nu beneficiaza de proprietatile lor. Singularity, in schimb, porneste de la premiza sigurantei limbajului si dezvolta o arhitectura de sistem care suporta si sporeste garantiile limbajului.

Singularity nu utilizeaza hardware pentru gestionarea memoriei pe procesoare pentru protectie, fapt care sugereaza posibilitatea reevaluarii acestui hardware. In general, multe programe se folosesc doar de putina functionalitate a hardware-ului ce se ocupa cu gestionarea memoriei. Sistemele „inglobate”(Embedded systems) se folosesc rareori de paginare, deoarece memoria este ieftina si abundenta. Spatiile de adrese largi(pe 64 de biti) reduc nevoia utilizarii spatiilor de adrese multiple pentru a obtine o limitare de 32 de biti. Singularity arata cum limbajele sigure si politica de partajare conservativa pot inlocui limitele de procese si inele de protectie, la un cost scazut. Hardware-ul curent, daca nu este utilizat integral, poate fi inlocuit de mecanisme mai simple cu mai putine blocaje de performanta, precum TLB-urile(TLB = Translation Lookaside Buffer – o memorie cache utilizata pentru a imbunatati viteza de translatie a adreselor virtuale).

Singularity poate beneficia de protectie a memoriei pentru „baza de incredere”. De exemplu, DMA in prezent este inerent nesigur si, datorita interfetelor diferite pe fiecare dispozitiv nu poate fi incapsulata sau virtualizata de un sistem. Protectia memoriei pentru transferurile DMA ar putea proteja sistemul impotriva DMA-ului directionat gresit. Suportul hardware pentru stivele segmentate poate reduce complexitatea compilatorului si executia mecanismului.

Garbage collector-ul este un serviciu automatic de gestiune a memoriei, si este implementat pentru a limita resursele cerute de o aplicatie. Este utilizat in foarte multe limbaje de programare deoarece permite programatorilor sa se concentreze pe continutul unei aplicatii si nu cum gestionare obiectele din cod.

Fiecare proces al sistemului de operare Singularity are propriul sistem de executie, cu memorie, algoritm de colectare a datei reziduale(garbage collector), si librarii proprii. Datorita independentei memoriei, un proces de executie poate fi customizat pentru a indeplini cerintele de implementare. In particular, colectorul de gunoi al proceselor poate fi selectat pentru algoritmul si structura sa de date, fara sa acorde atentie coordonarii cu omologii din alte procese.

4. Bibliografie

- *Microsoft Research* –

- *Sealing OS Processes to Improve Dependability and Security* - Galen Hunt, Mark Aiken, Paul Barham,etc.
  [1]***figura 1,*** [2]***figura 2***
  ftp://ftp.research.microsoft.com/pub/tr/TR-2006-51.pdf

- *Language Support for Fast and Reliable Messagebased Communication in Singularity OS* - Manuel F¨ahndrich, Mark Aiken, Chris Hawblitzel, Galen Hunt, Orion Hodson, etc.
  [5]**Planificatorul Rialto**
  http://www.cs.binghamton.edu/~kang/teaching/cs554/rialto96.pdf

- *An Overview of the Singularity Project* - Galen Hunt, James Larus, Martín Abadi, Mark Aiken, Paul Barham,etc.
  [4]**ABI**
  http://people.cs.vt.edu/~gback/qualifier/2006/TR-2005-135.pdf

- *Alte surse* –

- *Singularity (operating system)* – wikipedia.
  http://en.wikipedia.org/wiki/Singularity_%28operating_system%29

- *Strong and weak typing* – wikipedia.
  [3]**Definitie, corelatie.**
  http://en.wikipedia.org/wiki/Strong_and_weak_typing

- *Application binary interface* – wikipedia.
  [4]**Definitie**
  http://en.wikipedia.org/wiki/Application_binary_interface