Sisteme de operare avansate

Android – Gestiunea memoriei

Student: Bucurica Mihai Ionut

Master : anul I , IISC

Cuprins

1. Introducere Android

1.1. Scurt istoric

Android este o platformă software și un sistem de operare pentru dispozitive și telefoane mobile bazată pe nucleul Linux, dezvoltată inițial de compania Google, iar mai târziu de consorțiul comercial Open Handset Alliance. Android permite dezvoltatorilor să scrie cod gestionat în limbajul Java, controlând dispozitivul prin intermediul bibliotecilor Java dezvoltate de Google.

Lansarea platformei Android la 5 noiembrie 2007 a fost anunțată prin fondarea Open Handset Alliance, un consorțiu de 48 de companii de hardware, software și de telecomunicații, consacrat dezvoltării de standarde deschise pentru dispozitive mobile. Google a lansat cea mai mare parte a codului Android sub licența Apache, o licență de tip free-software și open source.
La 5 noiembrie 2007 a fost făcut public Open Handset Alliance, un consorțiu incluzând Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, Sprint Nextel și Nvidia, cu scopul de a dezvolta standarde deschise pentru dispozitive mobile. Odată cu formarea Open Handset Alliance, OHA a dezvăluit de asemenea primul său produs, Android, o platformă pentru dispozitive mobile construită pe nucleul Linux, versiunea 2.6.
La 9 decembrie 2008, a fost anunțat că 14 noi membri au aderat la proiectul Android, incluzând: Sony Ericsson, Vodafone Group Plc, ARM Holdings Plc, Asustek Computer Inc, Toshiba Corp și Garmin Ltd.
Începând cu 21 octombrie 2008, Android a fost disponibil ca Open Source. Google a deschis întregul cod sursă (inclusiv suportul pentru rețea și telefonie), care anterior era indisponibil, sub licența Apache. Sub licența Apache producătorii sunt liberi să adauge extensii proprietare, fără a le face disponibile comunității open source. În timp ce contribuțiile Google la această platformă se așteaptă să rămână open source, numărul versiunilor derivate ar putea exploda, folosind o varietate de licențe.

1.2. Caracteristici Android

Configurații dispozitive:	Platforma este adaptabilă la configurații mai mari, VGA, biblioteci grafice 2D, biblioteci grafice 3D bazate pe specificația **OpenGL ES 1.0** și configurații tradiționale smartphone.
Stocare de date	Software-ul de baze de date **SQLite** este utilizat în scopul stocării datelor
Conectivitate	Android suportă tehnologii de conectivitate incluzând GSM/EDGE, CDMA, EV-DO, UMTS, Bluetooth și Wi-Fi.
Mesagerie instant	SMS și MMS sunt formele de mesagerie instant disponibile, inclusiv conversații de mesaje text.
Navigatorul de web	Navigatorul de web disponibil în Android este bazat pe platforma de aplicații open source WebKit.

Mașina virtuală Dalvik	Software-ul scris în Java poate fi compilat în cod mașină Dalvik și executat de mașina virtuală Dalvik, care este o implementare specializată de mașină virtuală concepută pentru utilizarea în dispozitivele mobile, deși teoretic nu este o Mașină Virtuală Java standard.
Suport media	Android acceptă următoarele formate media audio/video/imagine: MPEG-4, H.264, MP3, AAC, OGG, AMR, JPEG, PNG, GIF.
Suport hardware adițional	Android poate utiliza camere video/foto, touchscreen, GPS, accelerometru, și grafică accelerată 3D.
Mediu de dezvoltare	Include un emulator de dispozitive, unelte de depanare, profilare de memorie și de performanță, un plug-in pentru mediul de dezvoltare Eclipse.
Piața Android	Similar cu App Store-ul de pe iPhone, Piața Android este un catalog de aplicații care pot fi descărcate și instalate pe hardware-ul țintă prin comunicație fără fir, fără a se utiliza un PC. Inițial au fost acceptate doar aplicații gratuite. Aplicații contra cost sunt disponibile pe Piața Android începând cu 19 februarie 2009
Multi-touch	Android are suport nativ pentru multi-touch, dar această funcționalitate este dezactivată (posibil pentru a se evita încălcarea brevetelor Apple pe tehnologia touch-screen).O modificare neoficială, care permite multi-touch a fost dezvoltată

1.3. Software Develpoment Kit

SDK-ul Android include un set complet de instrumente de dezvoltare. Acestea includ un program de depanare, biblioteci, un emulator de dispozitiv , documentație, mostre de cod și tutoriale. Platformele de dezvoltare sprijinite în prezent includ calculatoare bazate pe x86 care rulează Linux (orice distribuție Linux desktop modernă), Mac OS X 10.4.8 sau mai recent, Windows XP ,Vista, 7 sau 8. Cerințele includ, de asemenea, Java Development Kit, Apache Ant, și Python 2.2 sau o versiune ulterioară.

Mediul de dezvoltare (IDE) suportat oficial este Eclipse (3.2 sau mai recent), utilizând plug-in-ul Android Development Tools (ADT), deși dezvoltatorii pot folosi orice editor de text pentru a edita fișiere XML și Java și apoi să utilizeze unelte din linia de comandă pentru a crea, să construi și depana aplicații Android.

La 18 august 2008, a fost lansat Android SDK 0.9 beta. Această versiune oferă un API actualizată și extinsă, instrumente de dezvoltare îmbunătățite și un design actualizat pentru ecranul de bază. Instrucțiuni detaliate pentru actualizare sunt disponibile pentru cei care lucrează deja cu o versiune anterioară.

La 23 septembrie 2008 a fost lansat SDK-ul Android 1.0 . Conform documentației de lansare, includea "în principal remedii pentru probleme, deși au fost adaugate unele capabilități mai puțin semnificative". Includea, de asemenea, câteva modificări ale API-ului față de versiunea 0.9.

Pe 9 martie 2009, Google a lansat versiunea 1.1 pentru telefonul Android Dev. Deși există câteva actualizări estetice, câteva actualizări cruciale includ suport pentru "căutare prin voce, aplicații contra cost, remedii pentru ceasul cu alarmă, remediu pentru blocarea la trimiterea gmail, notificări de poștă electronică și intervale de împrospătare, iar acum hartile afișează evaluări de firme". Un alt update important este că telefoanele Dev pot acum accesa aplicații plătite și dezvoltatorii le pot vedea acum pe Piața Android.

1.4. Versiuni Android

Versiunile Android in ordine cronologica:

- Android beta , 5 noiembrie 2007
- Android 1.0 , 23 septembrie 2008
- Android 1.1 , 9 februarie 2009
- Android 1.5 – Cupcake, 30 aprilie 2009
- Android 1.6 – Donut , 15 septembrie 2009
- Android 2.0 /2.1 – Eclair, 26 octombrie 2009
- Android 2.2.x - Froyo, 20 mai 2010
- Android 2.3.x – Gingerbread, 6 decembrie 2010
- Android 3.x – Honeycomb, 22 februarie 2011
- Android 4.0.x – Ice Cream Sandwich , 19 octombrie 2011
- Android 4.1 – Jelly Bean, 27 iunie 2012
- Android 4.2 – Jelly Bean , 13 noiembrie 2012

2. Memoria RAM si memoria ROM

Legatura dintre CPU si memorie se realizeaza prin 3 magistrale:

- Magistrala de adrese ADRBUS;
- Magistrala de date DATABUS;
- Magistrala de control CONTROLBUS;

Magistrala de adrese este unidirectionala, cea de date este bidirectionala, iar cea de control contine linii de comanda si informatie de stare. In general, ca la orice calculator, citirea si scrierea memoriei se face paralel, deci se scrie sau se citeste un cuvant pe m biti. Adresarea memoriei se face prin magistrala de adrese si se presupune ca aceasta magistrala are n lunii. Atunci cand latimea cuvantului memorie nu este suficienta pentru reprezentarea datei, atunci se vor utiliza mai multe locatii consecutive. Daca de exemplu, o data se reprezinta pe doua cuvinte in memorie(UIA), atuci pentru a reprezenta data, se va folosi prima locatie pentru reprezentarea cuvantului mai putin semnificativ, iar locatia urmatoarea pentru cuvantul mai semnificativ(MSB). Procesoarele au facilitati de gestiune a memoriei. Operatiile de gestiune se refera la implementarea memoriei virtuale, protectia memoriei, etc.

Ca orice sistem de operare , Android are memorie ROM , unde sunt stocate toate informatiile sistemului de operare, si care nu se sterge niciodata si memorie RAM , unde sunt stocate date si care la restart se sterg.

La Android, dimensiunea minima a memorie ROM este de 95MB. Daca se incearca sa se micsoreze aceasta dimensiune sistemul poate deveni instabil si poate sa intre intr-o stare in care nu mai raspunde la comenzi.

2.1.Segmentarea

Segmentarea implică existenţa unei corespondenţe între adresa logică bidimensională definită de utilizator (numărul de segmente – s şi deplasarea în cadrul segmentului – d) şi adresa fizică unidimensională ce îi este asociată în memorie. Această corespondenţă este realizată prin intermediul tabelei de segment care conţine un număr de intrări egal cu numărul de segmente din programul respectiv. Fiecare intrare a tabelei conţine informaţii despre adresa de bază a segmentului şi despre dimensiunea acestuia şi poate fi accesată prin folosirea ca index a numărului de segment. Dacă deplasarea precizată în cadrul adresei logice nu satisface relaţia 0<d<dim, sistemul de operare va fi sesizat printr-o “capcană” cu semnificaţia „încercare de adresare în afară segmentului”. Dacă însă deplasare se încadrează între limitele permise, valoarea ei se adaugă la valoarea adresei de bază a segmentului, formându-se astfel adresa din memoria fizică a cuvântului dorit (se poate afirma deci, prin analogie, că tabela de segment este o colecţie de perechi de regiştri bază/limită).

Implementarea tabelei de segment

Soluţiile de implementare a tabelei de segment sunt asemănătoare celor folosite în cazul tabelei de pagină.

O primă variantă (preferată pentru tabelele de segment de mici dimensiuni) este utilizarea unor regiştri rapizi care să asigure o viteză de acces foarte mare şi, eventual, efectuarea simultană a operaţiilor de comparare a deplasării cu dimensiunea segmentului şi respectiv a însumării deplasării cu adresa de bază a segmentului.

Atunci când programul este format dintr-un număr mare de segmente se preferă ca tabela de segment să fie păstrată în memorie. Pentru a putea fi accesată, se foloseşte un registru bază al tabelei segment (RBTS) şi, deorece programele nu conţin acelaşi număr de segmente, se utilizează în plus un registru lungime al tabelei de segment (RLTS). În acest caz, pentru fiecare adresă logică (s,d) se verifică mai întâi corectitudinea numărului de segment (0<s<RLTS) şi apoi se însumează numărul segmentului cu conţinutul RBTS, rezultând astfel adresa din memorie a intrării tabelei de segment ce va fi citită. După aceea se procedează în mod obişnuit: se compară deplasarea cu dimensiunea segmentului şi se calculează adresa fizică a cuvântului dorit ca sumă a deplasării şi a adresei de bază a segmentului.

La fel ca şi în cazul paginării, cea dea doua variantă de implementare a tabelei de segment necesită două accese de memorie pentru o singură adresă logică, obligând sistemul de calcul să funcţioneze de două ori mai încet. Soluţionarea acestei probleme presupune folosirea unui set de regiştri asociativi în care să se păstreze cele mei recent utilizate intrări ale tabelei de segment.

2.2.Paginarea

Memoria RAM poate fi paginata. Ce inseamna paginarea? Inseamna ca sistemul de operare, in cazul nostru Android, imparte memoria in pagini(virtual). Acest lucru creste eficienta deoarece:

- Accesul la date se face foarte rapid: se stie exact din ce pagina sa se aduca informatia.
- Informatia este structurata mai usor.
- Paginile se pot actualiza dinamic pentru a tine in permanenta datele necesare .

In cazul in care se incearca accesul la o pagina care a fost in memoria RAM si nu mai este , apare eroarea de tipul : “page fault”. Eroarea de pagina apare cand programul incearca sa acceseze o pagina virtuala care nu este prezenta in memoria fizica. Orice intrare in tabela de pagini are in componenta sa un bit de absent/prezent(valid/invalid). El tine evidenta paginilor din spatiul adreselor virtuale care sunt prezente in memoria fizica. Astfel, daca acest bit are valoarea 0, inseamna ca pagina virtuala careia ii acoerspunde intrarea in tabela nu se afla in memoria fizica; o referinta la aceasta pagina va avea ca rezultat o capcana- eroarea de pagina.

Politici de inlocuire

Pentru acesti algoritmi, fiecarui proces i se aloca un numar fix de pagini de memorie la inceperea executiei, el nesolicitand spatiu de memorie suplimentar mai tarziu.

- **Algoritmul lui Belady de inlocuire optimala a paginilor (OPRA)**

Aceasta politica de inlocuire alege pentru eliminare cadrul care nu va fi folosit cea mai lunga perioada de timp (acela pentru care exista cel mai mare numar de instructiuni executate pana cand pagina noastra sa fie referita din nou). Pentru a implementa algoritmul ar trebui sa cunoastem numarul respectiv, ceea ce este imposibil in practica, neputand sa privim in viitor. De aici rezulta si inutilitatea sa intr-un sistem de operare real. Deorece, in teorie, acest algoritm reprezinta solutia optima, cu cea mai mica rata de erori (fault-rate), el este folosit ca un punct de reper pentru a masura performantele celorlalti algoritmi de inlocuire. Astfel, daca algoritmul lui Belady nu este cu mult superior ca performanta unui algoritm dat, se poate spune ca acesta din urma este destul de bun.

- **Random Replacement (Inlocuire Aleatoare)**

Dupa cum arata si numele, algoritmul alege pagina de eliminat in mod aleator, fiecare pagina avand sanse egale de a fi inlocuita. Astfel, nu mai trebuie sa tinem cont de sirul de referiri. Insa prezinta mari dezavantaje, ca natura sa imprevizibila, aparitia unui numar mare de erori de pagina si a fenomenului de trashing, despre care se va discuta mai tarziu.

Drept urmare, s-a renuntat la folosirea sa inca din anii ’60.

- **Algoritmul NRU (Not Recently Used- neutilizat recent)**

In acest algoritm un rol important il joaca bitii Referit/Modificat. Astfel, la inceperea unui proces, bitii R/M ai paginilor sale sunt resetati. La aparitia unei erori de pagina, bitul R(referit) este setat, se actualizeaza tabela de pagini si se restarteaza instructiunea. Pentru o modificare ulterioara asupra paginii respective, este setat bitul M. La un anumit interval de timp (o intrerupere de ceas, 20 ms), bitul R este resetat , pentru a tine evidenta paginilor care au fost referite recent. Folosindu-se de valorile bitilor R si M, algoritmul clasifica paginile din memorie in 4 categorii (clase):

- Clasa 0(R=0 -nereferita, M=0 - nemodificata);

- Clasa 1(R=0 -nereferita, M=1 - modificata);

- Clasa 2(R=1 -referita, M=0 - nemodificata);

- Clasa 3(R=1 -referita, M=1 - modificata);

Se observa ca in clasa 1 avem pagini nereferite, dar modificate. Acest lucru este posibil datorita resetarii periodice a bitului R, ceea ce muta o pagina din clasa 3 in clasa 1.

Algoritmul NRU alege pagina de inlocuit in mod aleator din cea mai joasa clasa ce contine elemente.Astfel este preferata o pagina din clasa 1(care a fost modificata, dar nereferita recent) in dauna uneia din clasa 2(nemodificata, insa folosita recent).

Ca avantaje, trebuie precizat ca acest algoritm este usor de implementat si inteles, oferind o adesea o performanta adecvata; un minus ar fi necesitatea parcurgerii bitilor R si M.

Algoritmul FIFO (first in, first out)

Acest algoritm este la fel de usor de implementat ca Random Replacement, insa, ca performanta, este cel mult la fel de bun. Functionarea sa se bazeaza pe mentinerea unei liste (o coada) a paginilor care se afla incarcate in memorie, cu proprietatea ca pagina incarcata de cel mai mult timp se afla pe prima pozitie, iar pagina incarcata cel mai recent pe ultima pozitie. La aparitia unei erori de pagina, se elimnia pagina de pe prima pozitie, iar noua pagina se incarca la capatul cozii. Se poate observa ca pentru implementare este suficient un singur pointer

Cum nu exista nici un fel de informatie care sa ne indice daca pagina cea mai veche din memorie este una folosita sau nu, exista posibilitatea de a inlatura o pagina referita frecvent, cea ce reprezinta un minus pentru algoritm. Astfel, este folosit rar in practica in aceasta forma.

FIFO are o proprietate negativa: sufera de anomalia lui Belady. Acest lucru inseamna ca, la cresterea numarului de cadre de pagina, exista posibilitatea de crestere a numarului de erori de pagina.

- **Algoritmul LRU (Least Recently Used= Cel mai putin recent utilizat)**

Acest algoritm este o aproximare a algoritmului optimal al lui Belady; el porneste de la ideea ca paginile ce nu au mai fost folosite de mult timp au sanse mari sa nu fie folosite nici de acum inainte, de aceea ele capata prioritate cand apare o eroare de pagina si trebuie eliberat un cadru (se alege pagina care nu a mai fost referita de cel mai mult timp).

2.3. Paginarea segmentelor

Atât paginarea cât şi segmentarea au avantaje şi dezavantaje. Acesta este motivul pentru care, uneori, s-a preferat combinarea celor două metode de gestionare a memoriei prezentate anterior.

Un exemplu în acest sens este şi paginarea segmentelor , în cadrul căreia se foloseşte avantajul eliminării fragmentării externe oferit de către paginare (pentru alocare poate fi folosit oricare dintre cadrele disponibile). Deosebirea dintre această metodă şi segmentare este aceea că intrarea tabelei de segment nu conţine adresa de bază a segmentului ci adresa de bază a unei tabele de pagină asociată acestui segment. Segmentul este format dintr-un număr de pagini de

aceeaşi dimensiune. Deplasarea în cadrul segmentului se exprimă prin număr de pagină şi deplasare în pagină. Cu numărul de pagină folosit ca index în tabela de pagină a segmentului, se obţine numărul cadrului care, în final, se combină cu deplasarea în pagină şi formează adresa fizică. Deoarece ultima pagină a fiecărui segment nu este întotdeauna complet ocupată, metoda introduce totuşi o cantitate redusă de fragmentare externă (în realitate, atunci când se lucrează cu tabele de segment de dimensiuni mari, mecanismul este ceva mai complicat).

3. Memory Management Unit- MMU

MMU la Android funcționează cu sistemul de memorie cache pentru a controla accesul la și de la memoria externă. MMU controlează, de asemenea, traducerea adreselor virtuale in adrese fizice.

Procesorul ARM pune în aplicare un MMU ARMv6 de a furniza translatarea de adrese si permisiunea de acces si control pentru porturile de instructiuni și de date ale procesorului. MMU controlează hardware-ul de tabela de control care accesează tabelele de translatare în memoria principală. Un singur set de tabele de pagini cu două nivel stocate în memoria principală controlează conținutul de instruire și secundare de date *Translation Lookaside Buffers* (TLBs). Traducerea adresei fizice in adresa virtuala este pusa în TLB(Translation Lookaside Buffer).

Caracteristicile MMU sunt:

- Dimensiuni standard de mapare, ale domenilor si scheme de protectie si acces.
- Dimensiunile de mapare sunt: 4KB, 64KB, 1MB,16MB.
- Permisiunea de acces pentru sectiunile de 1 MB SI 16 MB, sunt specificate pentru intreaga sectiune.
- aveți posibilitatea să specificați permisiuni de acces pentru 64KB pagini mari și paginile de 4KB mici separat pentru fiecare trimestru al paginii (aceste sferturi sunt numite subpagini)
- 16 domenii
- aveți posibilitatea să marcați intrările ca o cartografiere la nivel mondial, sau asociate cu un identificator specific spațiu cerere pentru a elimina cerința de bufeuri de TLB pe switch-uri cele mai multe context
- permisiunile de acces extins pentru a permite supraveghetorului doar în citire și supervizor / utilizator read-only moduri de a fi în acelaşi timp sprijinite
- atribute regiune de memorie, pentru a marca paginile partajate de mai multe procesoare
- tabelul de pagină
- algoritmul de inlocuire Round-Robin

Arhitectura MMU a memoriei de sistem permite granulatie fina de control al unui sistem de memorie. Acest lucru este controlat de un set de mapări virtuale la adresa fizică și proprietățile asociate memorie deținute în cadrul uneia sau mai multor structuri cunoscute ca TLBs în cadrul MMU. Conținutul TLBs sunt gestionate prin intermediul „hardware translation lookups" dintr-un set de tabele de traducere în memorie.

Pentru a preveni necesitatea unei invalidari a TLB pe un context de comutare, puteți marca fiecare translatare de la adresa virtuala la cea fizica ca fiind asociate cu un spațiu aplicație special, sau ca la nivel global pentru toate spațiile de aplicații. Doar mapările globale și cele pentru spațiul de cererea curentă sunt activate în orice moment. Prin schimbarea *Application Space IDentifier* (ASID) puteți modifica setul permis de mapări virtuale de adrese fizice.

3.1. TLB

TLB este o memorie de tip cache (foarte rapida, care asigura in permanenta procesorului datele de care are nevoie , pentru a evita pierderea timpului cu cautarea in memoria RAM) pe care MMU o utilizeaza pentru a imbunatati viteza de translatie a adreselor virtuale. TLB este implementat sa semene cu un CAM(content-addresable memory – sau in traducere: continut adresabil de memorie). Cheia de cautarea CAM este adresa virtuala si rezultatul cautarii este adresa fizica. Daca adresa ceruta este prezenta in TLB , atunci CAM returneaza rezultatul imediat si adresa fizica cautata poate fi folostia pentru a accesa memoria. Daca nu se gaseste adresa , atunci se cauta in toata memoria RAM(proces in care se pierde timp), si cand se gaseste se returneaza adresa fizica, si se mapeaza si in TLB.

TLB are nu numar fix de sloturi care contin intrari ale paginilor in tabel, care mapeaza adresa virtuala de adresa fizica. Memoria virtuala este cea care este vazuta intr-un proces. Acest spatiu virtual este impartit in pagini de o anumita marime. Putem spune ca TLB este de fapt un cache al tabelului de pagini.

TLB se gaseste intre CPU si CPU cache, intre CPU cache si memoria RAM, sau intre nivelurile memoriei cache. Locul in care se gaseste determina modul in care se face adresarea. Daca cache-ul este adresat virtual atunci cererile sunt trimise direct de la CPU catre cache, iar TLB este accesat doar in cazul unui "miss". Daca cache-ul este adresat fizic atunci CPU face o cautare in TLB si returneaza adresa fizica care este trimisa la CPU.

3.2.Memory access sequence(Secventa de acces la memorie)

Cand procesorul genereaza un acces la memorie, MMU cauta maparea la adresa virtuala si tabela. Daca nu se gaseste , atunci MMU creeaza o mapare. Daca se gaseste maparea, atunci se verifica bitii de permisiune si de domeniu sa se verifice daca se poate acorda accesul. Daca nu se acorda permisiunea, MMU semnaleaza "memory abort". Fiecare intrare in TLB contine o adresa virtuala, o marime de pagina, o adresa fizica si un set de proprietati de memorie. Fiecare este marcat ca fiind asociat cu o aplicatie particulara sau cu mai multe. Registrul c13 in CP15 determina spatiul. O intrare TLB se potriveste daca bitii 31:N se potrivesc, unde N este $\log_2$ din marimea paginii in header.

3.3. Pornirea si oprirea MMU.

Se poate opri si porni MMU modificand bitul c1 din CP15. Inainte de a porni MMU, trebuie:

- Programat toti registrii CP15 relevanti.
- Opri si invalida Cache-ul de instructiuni.
- Programa *"Translation Table Base si Domain Access Control Registers."*
- Programa primul si al doilea nivel de descriptori.

3.4.MMU aborts

Mecanisme care pot cauza procesorului de a lua o excepție din cauza unui acces de memorie sunt:

- **Debug abort:** Se detecteaza un breakpoint sau un watchpoint
- **MMU fault:** MMU detectează o restricție și semnalizează procesorul.
- **External abort:** Memoria externa detecteaza un acces ilegal sau intrerupt la memorie.

Toate se incadreaza in genericul „aborts". Daca o cerere de acces la memorie este anulata si este o instructiune de tip „fetch", atunci se apeleaza exceptia" Prefetch". Daca anularea este de tipul data access sau operatiune de mentenanta de cache atunci se apeleaza „Data Abort" si se si inregistreaza eroarea in „Data Fault Status Register" pentru a se determina cauza erorii.

Erorile de memorie externa sunt cele care se intampla in memoria sistemului altele decat cele detectate de MMU. In general sunt foarte rare si sunt fatale procesului in desfasurare. Un exemplu de un astfel de eveniment este un bit de paritate care nu poate fi corectat in timpul ECC la o memorie structurata pe 2 nivele.

Erorile de memorie externe sunt definite ca cele care apar în sistemul de memorie, altele decât cele care sunt detectate de un MMU. Erorile de memorie externe sunt de așteptat să fie extrem de rare și sunt susceptibile de a fi fatal pentru procesul de funcționare. Un exemplu de eveniment care poate provoca o eroare de memorie extern este o paritate nerectificabile sau eșecul ECC pe o nivelul doi de memorie structură.

3.5.MMU fault checking

În timpul prelucrării de o secțiune sau pagină, MMU se comporta diferit, pentru că face verificarea defectelor.MMU generează patru tipuri de defecte:

- Defect de aliniament
- Defect de translatare
- Defect de acces la flag
- Defect de domeniu
- Defect de permisiune

Erorile care sunt detectate de catre MMU sunt tratate inainte de orice altfel de defect sau eroare. Verificarea de erori de aliniament poate fi oprita sau pornita modificand bitul A din Registrul de Control.De asemenea este independent de faptul ca MMU este oprit sau pornit. Celelalte erori sunt tratate doar cand MMU este pornit. Mecanismul de control de acces al MMU detecteaza conditiile care produc aceste erori. Daca o eroare este detectata ca rezultat al accesului la memorie, MMU anuleaza accesul si semnaleaza procesorului acest lucru. MMU retine statusul si informatie despre adresa.

3.6.MMU- descriptorii

Pentru a sprijini paginarea se utilizează un descriptor pe două niveluri de definiție. Descriptorul de primul nivel indică dacă accesul este la o secțiune sau într-o pagina de tabel. În cazul în care accesul este la un tabel pagină, se determina tipul si aduce un descriptor de nivel 2.

4. Dalvik Virtual Machine

4.1. Introducere

Managementul memoriei si al procesorului in Android este un pic diferit. Android foloseste propria masina virtuala si propriul “run-time” pentru a gestiona aplicatiile de memorie. De asemenea gestioneaza si timpul de viata al proceselor. Android asigura raspunsul aplicatiilor prin oprind procese care nu mai sunt necesare pentru a face rost de resurse pentru aplicatiile cu prioritate mai ridicata.

Fiecare proces ruleaza intr-un proces separat cu o instanta Dalvik unica. Dalvik si “run-time-ul” Androidului sunt cap de lista al kernelelor Linux care folosesc hardware low-level, incluzand drivere si managementul memoriei.

4.2. Dalvik Virtual Machine

Este o masina virtuala bazata pe registrii, care a fost optimizata pentru a asigura ca un “device” poate rula mai multe instante cu succes in acelasi timp.Se bazeaza pe kernelul Linux pentru “threading” si managementul memoriei de nivel scazut. Asigura deasemenea siguranta, securitate si protectie a datelor.Toate hardware Android folosesc Dalvik ca nivel de mijloc. Folosind o masina virtuala pentru a gazdui executarea aplicatiilor, dezvoltatorii au un nivel de abstractizare care asigura ca nu trebuie sa isi faca griji de o implementare particulara a vreunui hardware.

Masina virutala Dalvik utilizeaza fisiere executabile care au un format optimizat pentru a asigura cerinta minima la memorie. Executabilele de tip .dex sunt create prin transformarea limbajului Java de catre SDK.

4.3.Procesele in Android

Ordinea in care procesele sunt oprite si distruse este determinata de prioritatea aplicatiilor hostate. Prioritatea unei aplicatii este egala cu prioritatea componentei care ruleaza serviciul.

Daca doua aplicatii au acelasi grad de prioritate, procesul care a fost la un nivel de prioritate mai mica o perioada mai indelungata va fi distrus primul. Nivelul de prioritate este deasemenea afectat de dependintele inter-proces. Daca o aplicatie are dependinte la un anumit serviciu , atunci aceea aplicatie va avea cel putin nivelul de prioritate ca serviciul pe care il determina.

Toate procesele vor ramane in memorie pana cand sistemul decide ca trebuie sterse pentru a face loc altor aplicatii.

Tipul proceselor:

1. **Active Processes** – sunt cele care gazduiesc aplicatii cu componente care interactioneaza la acel moment cu utilizatorul. Aceste procese sunt cele pe care Android incearca sa le faca cat mai “responsive” . In general sunt foarte putine aceste procese si vor fi distruse ultimele.
2. **Visible Processes –** sunt inactive , dar care gazduiesc activitatile vizibile.
3. **Started Service Processes** – sunt procese care gazduiesc servicii care au inceput. Serviciile suporta procesele care ruleaza dar care nu sunt visibile in interfata. Pentru ca serviciile nu interactioneaza in mod direct cu utilizatorul, au un nivel de prioritate mai scazut, dar sunt inca vazute ca fiind esentiale.

4. **Background Processes** – sunt procese care gazduiesc activitati care nu sunt visibile si nici nu folosesc vreun serviciu. In general sunt in numar mare, si pot fi cu usurinta distruse de sistem.
5. **Empty Processe** - sunt procese retinute in memorie pentru a optimiza performantele globale.

5. Bibliografie

[1] http://infocenter.arm.com/help/topic/com.arm.doc.ddi0211i/DDI0211.pdf
[2] http://source.android.com/tech/debugging/native-memory.html
[3] http://mobworld.wordpress.com/2010/07/05/memory-management-in-android/
[4] http://support.google.com/android/bin/answer.py?hl=en&answer=168609
[5] http://en.wikipedia.org/wiki/Page_replacement_algorithm
[6] http://en.wikipedia.org/wiki/Paging
[7]http://www.elcom.pub.ro/discipline/amp2/curs/Elemente%20de%20gestiune%20a%20memoriei_1_s.pdf