Universitatea din București

Facultatea de Matematică și Informatică

Specializare: Calculatoare și Tehnologia Informației

Proiect utilizarea sistemelor de operare

Profesor coordonator

Drăgan Mihăiță

Studenți

Toader Andrei-Pavel

Andriță Lucian-Gabriel

București 2020

Universitatea din București

Facultatea de Matematică și Informatică

Specializare: Calculatoare și Tehnologia Informației

Watchdog

Profesor coordonator

Drăgan Mihăiță

Studenți

Toader Andrei-Pavel

Andriță Lucian-Gabriel

București 2020

***Cuprins***

1) Introducere.............................................................................................................................1

*1.1) Generalități și mod de funcționare.................................................................................1*

*1.2) Aplicabilități...................................................................................................................2*

*1.3) Comparație între dispozitivul încoporat și cel extern......................................................3*

2) Tehnologii utilizate................................................................................................................4

3) Descrierea codului..................................................................................................................5

*3.1) Preluarea datelor...........................................................................................................5*

*3.2) Prelucrarea datelor........................................................................................................7*

*3.3) Verificarea stării proceselor..........................................................................................9*

Concluzii..................................................................................................................................11

Bibliografie...............................................................................................................................12

***1) Introducere***

1.1) Generalități și mod de funcționare

Un dispozitiv de tip “câine de pază” (watchdog) este un dispozitiv folosit pentru a proteja un sistem de erori atât software, cât și hardware, ce pot provoca blocări ale sistemului sau chiar încetarea activității acestuia. În mare, dispozitivul se bazează pe primirea unor semnale de la senzorii ce urmăresc componentele cât și de la aplicații ce supraveghează partea de software la un interval de tip prestabilit. Astfel asigură calitatea funcționării programului și implicit lipsa erorilor de orice fel. În caz contrar, acesta va efectua diferite comenzi cu scopul de a reporni sistemul cu totul sau aplicația care a încetat să răspundă.

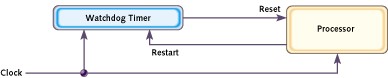


Figura 1. Structură generală [1]

Dispozitivul poate fi atât extern (cronometrul având un ceas separat), cât și intern (Figura 1-având același ceas). Modul de funcționare va rămâne același: atât procesorul cât și cronometrul de tip watchdog vor depinde de același semnal de intrare (ceasul), iar atunci când cronometrul va ajunge la 0 înainte să primească semnalul de restart (repornind astefel numărătoarea) din partea procesorului (adică există o eroare în sistem), dispozitivul va reseta sistemul sau doar aplicația/procesul cu pricina.

Figura 2 exemplifică mai bine modul de funcționare descris în paragraful anterior, făcând referire la cazul favorabil în care sistemul nu are erori, fiind alimentarea dispozitivului (ce va rămâne constantă), iar fiind resetarea sistemului (care nu se va întâmpla). Înainte ca numărătoarea (WDT TIME-OUT) să ajungă la 0, se va aștepta o perioadă de ceas pentru a verifica primirea (sau lipsa) semnalului de restart (INTERNAL RESET). Atunci când va trece perioada de ceas, iar semnalul a fost primit, se execută reinițializarea cronometrului (RESET TIME-OUT), proces ce este variabil și stabilit în funcție de necesități ().

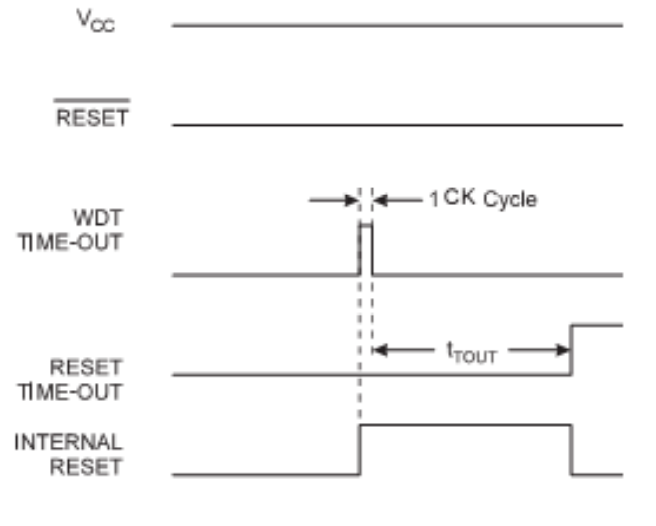
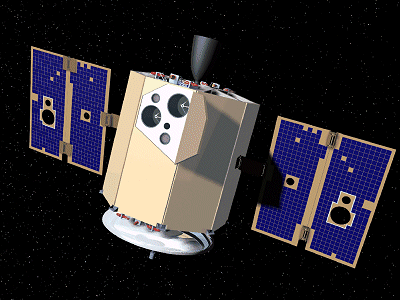


Figura 2. Diagrama temporală a dispozitivului [2]

1.2) Aplicabilități

Dispozitivele de tip watchdog sunt folosite în special în sistemele ce operează la distanță (precum sateliți, sonde de minerit etc), ale căror resetare trebuie făcută într-o anumită fereastră de timp, astfel timpul de reacție uman fiind depășit din diferite motive: imposibilitatea de a identifica eroarea în timp util, durata trimiterii instrucțiunilor (lucrul în spațiu). Din aceste cauze, este necesar un sistem local (preferabil extern sistemului pentru a evita orice posibilitate de corupere a codului sau chiar nefuncționarea acestuia) cu scopul de a supraveghea și menține sistemul într-o stare optimă de funcționare.

Un exemplu în care un dispozitiv de această natură nu a fost implementat, în ciuda necesității sale, este misiunea Clementine. Aceasta a fost o colaborare dintre Administrația Națională Aeronautică și Spațială (National Aeronautics and Space Administration-NASA) și Organizația de Apărare a Rachetelor Balistice (Ballistic Missile Defense Organization-BMDO) cu scopul de a testa senzori și componente în condiții spațiale și de a colecta date despre Lună și un asteroid din vecinătatea Pământului. Dupa strângerea datelor de pe Lună, sonda a întâmpinat o eroare de software care a dus la pornirea și blocarea sistemelor de propulsie. Controlorii de la sol au reușit după 20 de minute să reseteze softwareul sondei. Aceasta era în stare funcțională și transmitea date, dar din Figura 3. Sonda Clementine [3]

cauza erorii a ars tot combustibilul de la bord, astfel renunțându-se la apropierea și inspectarea asteroidului.

1.3) Comparație între dispozitivul încoporat și cel extern

Watchdogul intern este folosit în special pentru sistemele cu un buget redus, oferind o protecție minimală împotriva codului corupt, existând posibilitatea unei reprogramări a dispozitivului în anumite circumstanțe. Această posibilitate este datorată existenței sale în procesor, folosirea la comun a resurselor și utilizarea aceluiași semnal de control (clock). Efectul coruperii codului poate duce la o resetare necorespunzătoare a sistemului și implicit la blocarea acestuia.

Watchdogul extern este mai costisitor de implementat, dar atunci când este construit eficient (luând în calcul fiecare scenariu posibil) devine imposibil de dezactivat de către software. Această posibilitate este datorată existenței sale independente de sistem, astfel poate fi conectat la diferite circuite integrate (ICs) care necesită o resetare pentru o funcționare optimă.

***2) Tehnologii utilizate***

Limbajul C este un limbaj de programare de nivel mediu dezvoltat de Dennis Ritchie la începutul anilor '70, în timp ce lucra la American Telephone & Telegraph (AT&T) Bell Labs în Statele Unite al Americii. Obiectivul dezvoltării sale a fost în contextul reproiectării sistemului de operare UNIX pentru a permite utilizarea acestuia pe mai multe calculatoare.

Anterior, limbajul de programare B era folosit pentru îmbunătățirea sistemului UNIX deoarece a permis o scriere de cod mult mai rapidă decât în ​​limbajul de asamblare. Cu toate acestea, limbajul B avea și dezavantaje deoarece nu înțelegea tipurile de date și nu asigura utilizarea „structurilor”. Dennis Ritchie a păstrat cea mai mare parte a sintaxei limbajului B și a adăugat tipuri de date și multe alte modificări necesare.

Limbajul de programare C a fost dezvoltat în perioada 1971-1973, conținând atât funcționalități noi, cât și caracteristici detaliate necesare programării unui sistem de operare. Prin urmare, multe dintre componentele UNIX, inclusiv nucleul UNIX în sine, au fost în cele din urmă rescrise în limbajul C.

***3) Descrierea codului***

Codul se bazează pe limbajul C apelând ulterior funcțiile din terminal pentru a strânge informațiile necesare execuției acestuia. În primul rând primim drept date de intrare lista cu procesele pe care acesta le va urmări și reglementa.

La un interval prestabilit, rulăm programul automat, odată cu pornirea echipamentului la un interval stabilit folosind daemonul cron (se va insera în fișier \*/x\*\*\*\*/path, unde x reprezintă timpul în minute, iar path este pathul absolut către executabil) și adunăm astfel datele într-un fișier de tip .txt.

Acesta din urmă va fi segmentat în 4 fișiere individuale, fiecare stocând un tip de informație [identificator de proces (process identifier-PID), dimensiunea procesului (resident size-RES), utilizarea procesorului (cpu usage-%CPU), utilizarea memoriei (memory usage-%MEM)]. Vom folosi tablourile unidimensionale pentru identificarea proceselor după PID și vom reglementa resursele folosite (RES, %CPU, %MEM).

**3.1) Preluarea datelor**

Fișierul lista.txt va conține PIDurile proceselor care trebuie urmărite și limitele aferente acestora. PIDurile vor fi stocate ulterior de către vectorul “v” alocat dinamic în funcție de numărul de procese, iar limitele, asemenea PIDurilor vor fi păstrate în tablouri uni/bidimensionale. Informațiile despre procesele curente vor fi afișate în fișierul procese.txt aflat în directorul Fisiere\_suport.

Pentru a urmări cu ușurință modificările făcute de aplicație, vom avea un fișier numit registru.txt deschis în modul “append” în care se vor afișa datele preluate despre procese și schimbările aduse acestora. La începutul fiecărei intrări în jurnal se vor scrie data și ora executării pentru a facilita o depanare ulterioară.



Figura 4. Stocare informații de la utilizator

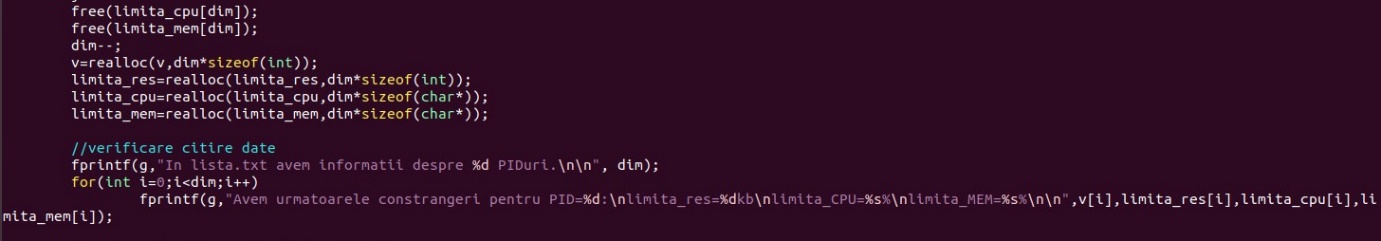


Figura 5. Verificare integritate informații

În funcția system() care permite încorporare comenzilor din terminal în limbajul C, folosim comanda top() cu opțiunile –b –n 1 pentru a opri rularea continuă a funcției la o singură instanță, iar variabila LUNG va reține numărul de linii din fișierul procese.txt pentru a facilita parcurgerea elementelor. În continuare folosindu-ne de comanda awk() (limbaj de programare pentru manipularea datelor și generarea rapoartelor; este o abreviere ce conține inițialele celor 3 inventatori: Aho, Weinberger și Kernighan). Vom prelua coloanele 1, 6, 9 și 10 corespunzătoare pentru PID, RES, %CPU, %MEM și le vom introduce în fișierele aferente.

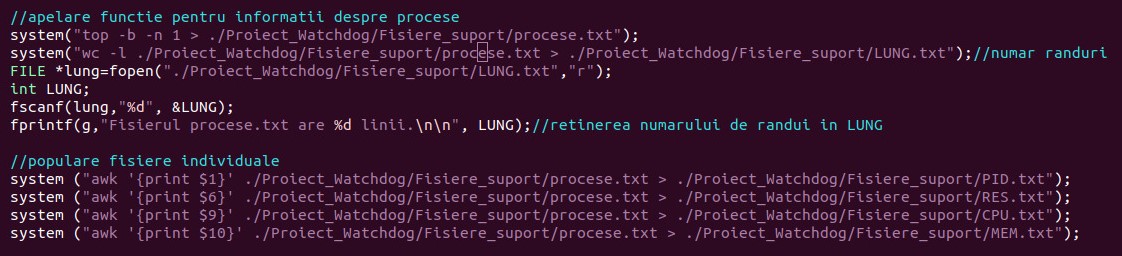


Figura 6. Segmentare informații

**3.2) Prelucrarea datelor**

Se va deschide fiecare fișier în care au fost preluate datele, apoi urmează stocarea acestora. PIDurile și RESurile vor fi păstrate în tablouri unidimensionale de tip întreg, iar resursele în tablouri bidimensionale de tip char deoarece valorile reale sunt interpretate ca fiind șiruri de caractere din cauza virgulei.

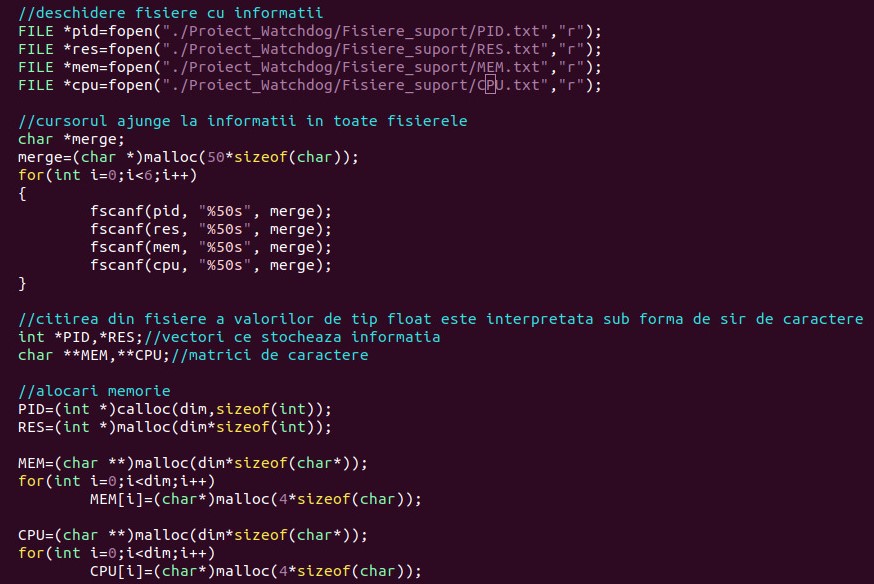


Figura 7. Stocare informațiilor oferite de funcția top()

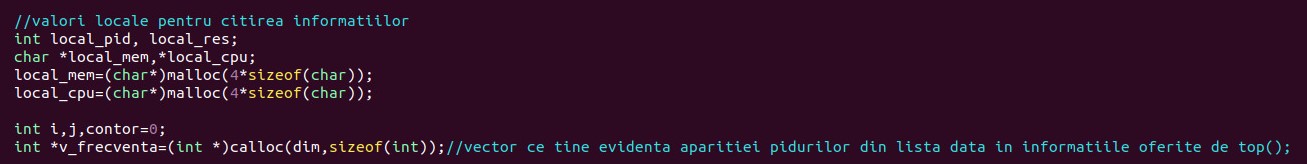
Folosim un set de variabile locale pentru citirea efectivă a informațiilor din fișiere, fiecare având tipul de date corespunzător, iar v\_frecvență va fi un vector de frecvență ce va ține evidența apariției proceselor din lista dată în procesele preluate de funcția top() de pe echipament.

Figura 8. Declarări variabile locale

În Figura 9 se parcurg simultan fișierele (pid, res, mem, cpu) împreună cu lista de PIDuri (vectorul v) pentru a verifica existența proceselor care trebuie urmărite și formarea tablourilor unidimensionale și bidimensionale cu informațiile despre fiecare proces.

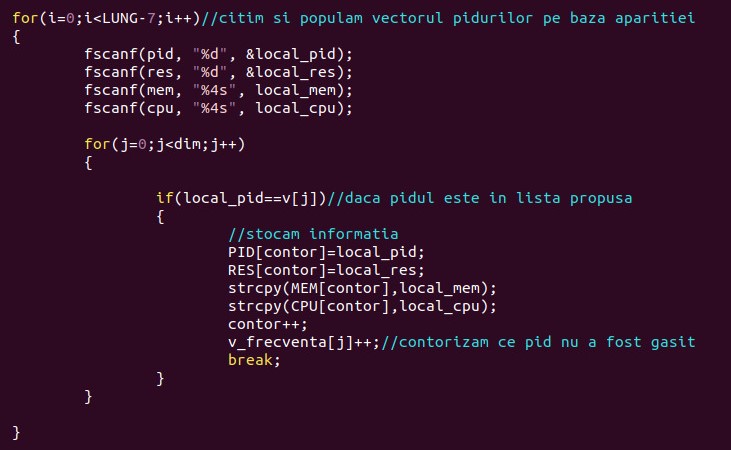


Figura 9. Confirmarea existenței procesului

**3.3) Verficarea stării proceselor**

Într-un prim pas verificăm prin intermediul PIDului dacă procesele menționate în listă sunt prezente. Folosirea comenzilor din terminal în funcție de PIDuri ne impune să folosim funcția sprintf() pentru a compune textul comenzii și a facilita apelarea ei.

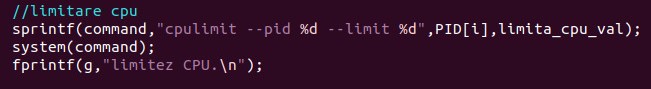


Figura 10. Apelarea variabilă a comenzilor din terminal

Dacă sunt prezente, vom începe sistematic compararea utilizării curente a resurselor de către procese cu limitele impuse de utilizator, o excepție fiind la valorile pentru %CPU și %MEM unde s-a făcut o conversie de la șir de caractere la float prin funcția atof() pentru a ne asigura de funcționarea aplicației și pentru valori de 2 cifre. Folosirea comenzilor din terminal în funcție de PIDuri ne impune să folosim funcția sprintf() pentru compune textul comenzii și a facilita apelarea ei.

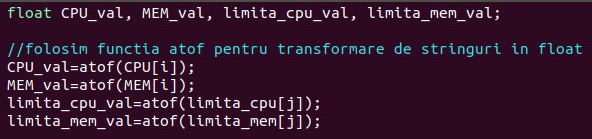


Figura 11. Transformare date

Mai întâi vom verifica dacă dimensiunea procesului curent se încadrează în limita stabilită, iar în caz contrar se va închide procesul deoarece RES ca resursă nu este una dinamică, adică nu își va schimba valoarea indiferent de circumstanțe.

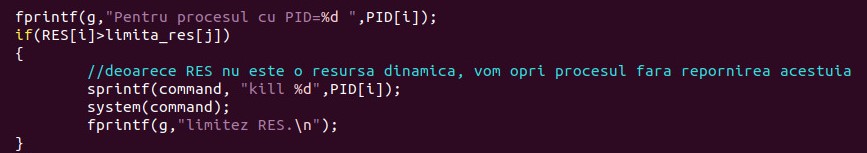


Figura 12. Limitare pe baza RES

Următoarea verificare va avea în vedere funcționarea procesului, cazul contrar fiind lipsa utilizării procesorului (CPU\_val==0). Folosim comanda cat() pentru a lua pathul către executabil aflat în /proc/[PID]/cmdline. După ce acesta este stocat, închidem procesul, iar mai apoi apelăm pathul către executabil.



Figura 13. Secvența de repornire a proceselor

Ulterior verificăm procentajul de utilizare al procesorului și a memoriei pentru fiecare proces și vom impune limitele date de utilizator prin comenzile:

* cpulimit() cu opțiunile - -pid (pentru identificarea procesului) și - -limit (pentru impunerea limitei procesorului)
* niceload() cu opțiunea - -rm (pentru impunerea limitei memoriei)

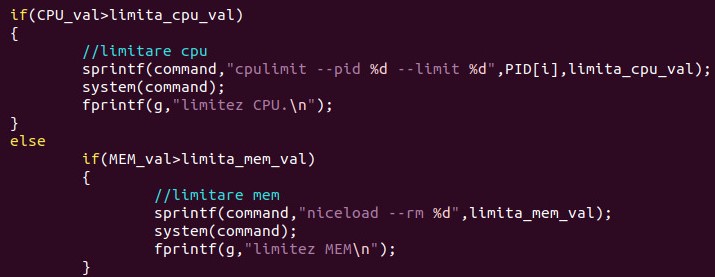


Figura 14. Limitare resurse dinamice

***Concluzii***

Watchdogul este un dispozitiv necesar în situația în care avem un sistem independent, a cărei funcționare nu implică intervenția umană, rolul acestuia fiind unul vital: de supraveghere și menținere în parametrii optimi de funcționare. În urma dezvoltării părții de software a acestuia, am descoperit complexitatea rolului său menționat mai sus, deoarece a fost nevoie de o analiză a tuturor proceselor în timp real ale sistemului și compararea datelor cu cele oferite de utilizator.

Implementarea watchdogului în limbajul de programare C a prezentat diferite probleme deoarece acesta trebuia să comunice cu sistemul de operare pentru a primi informații despre procese și pentru a le manipula. Aici a intervenit foarte mult comanda system() care ne-a ajutat să facem o punte între program și terminal.

***Bibliografie***

Surse informații:

1. <https://www.webopedia.com/TERM/W/watchdog.html>
2. <https://www.embedded.com/introduction-to-watchdog-timers/>
3. <https://os.mbed.com/cookbook/WatchDog-Timer>
4. <http://www.ganssle.com/watchdogs.pdf>
5. <https://www.maximintegrated.com/en/design/technical-documents/app-notes/4/4229.html>
6. <https://www.nasa.gov/mission_pages/LCROSS/searchforwater/clementine.html>
7. <https://www.geeksforgeeks.org/benefits-c-language-programming-languages/>

Surse poze:

1. <https://www.embedded.com/introduction-to-watchdog-timers/>
2. <http://courses.daiict.ac.in/pluginfile.php/8702/mod_resource/content/0/watchdog.pdf>
3. <https://www.nasa.gov/mission_pages/LCROSS/searchforwater/clementine.html>