PalmPilot™에서의 Disconnected Swapping에 관한 연구

손정현, 이재원, 윤현수
한국과학기술원 전산학과
jhsohn@camars.kaist.ac.kr

Disconnected Swapping on PalmPilot™

Jhsohn Sohn, Jaewon Lee, Hyonsoo Yoon
Dept of Computer Science, KAIST

최근 들어 수요와 공급이 급격히 증가하고 있는 PDA(Personal Digital Assistants)의 운영체제는 기존에 많이 연구되어 왔던 운영체제 환경과는 하드웨어의 특성상 메모리 관리나 파일 시스템 등에서 많은 차이점을 보인다. 예를 들어 작은 메모리 용량과 하드디스크 없이 메인 메모리만을 사용해서 모든 데이터를 저장해야 하므로 이를 얼마나 효율적으로 관리해야 하는지가 문제점이 된다.
여기서 우리는 PDA 중에서 비교적 많은 사용자 층을 갖고 있는 USRobotics 사의 PalmPilot [1] 을 대상으로 도킹 데스크탑을 이용한 swapping 방법을 모색해 보았다.
기존의 PalmPilot은 그림 1과 같이 데스크탑과 연결하여 자료의 백업과 동기화 작업만을 수행하였다. 하지만 이 순간 만큼은 PalmPilot도 일반 데스크탑과 같이 대용량의 하드디스크를 사용할 수 있는 기회가 생기므로 이 때를 잘 이용한다면 사용자는 좀 더 커다란 데이터를 사용할 수 있게 될 것이다.

1. 서 론

PalmPilot은 1 Mbyte 정도의 SRAM만을 이용하여 모든 응용 프로그램과 데이터를 저장해야 한다. 만일 메인 메모리의 용량을 초과하여 데이터를 저장한다면 시스템은 심한 경우 down 되어 reset을 해야 하는 지경에 이르기까지 한다.
만일 일반 데스크탑과 같이 하드디스크를 이용하여 swapping을 할 수 있다면 이러한 일을 겪지 않을 것이며 사용자는 메모리의 제한을 덜 받게 될 것이다.
우리는 PalmPilot이 도킹 데스크탑에 Cradle[1] 을 이용해 연결되었을 때 swapping을 함으로써 사용자가 메모리의 제약을 덜 받도록 하는 방법을 연구하였다. 잘 사용되지 않는 메모리 영역은 데스크탑의 디스크에 저장시켜 놓고 PalmPilot의 메모리에선 지움으로써 사용자가 보다 많은 메모리를 사용할 수 있도록 한다.
그러기 위해서는 일반적인 swapping 메커니즘과는 다른 방법을 사용해야 한다. 즉, PalmPilot 사용자는 평소에는 도킹 데스크탑과는 완전히 분리된 상태에서 작업을 하므로 이 때는 swapping을 전혀 할 수 없으며, 이미 하드디스크로 swap-out된 데이터는 Cradle에 연결해 데스크탑으로 부터 다시 읽어오기 전에는 사용할 수 없다.

[1] PalmPilot을 시리얼 포트를 이용해 데스크탑 PC와 연결시켜주는 장치

우리는 우선 PalmPilot의 메모리 아키텍처에 대해 설명하여 PalmPilot에 대한 이해를 도운 뒤, 새롭게 제시되는 swapping 메커니즘에 대해 설명하게 될 것이다. 이를 통해 PalmPilot 을 도킹 데스크탑에 연결했을 때 효율적인 swapping으로 사용자가 사용할 수 있는 충분한 메인 메모리를 항상 확보할 수 있는 방법을 설명할 것이다.

2. PalmPilot 메모리 아키텍처

PalmPilot의 CPU는 Motorola 68328으로써 32-bit address를 사용한다.[5][6] 따라서 전체 memory address space는 4GB까지 사용할 수 있지만, 실제적으로는 보통 1MB 정도의 RAM과 512KB 정도의 ROM을 사용하고 있다.
이들 RAM과 ROM은 하나의 메모리 카드안에 같이 들어가 있고, 메모리 카드는 PalmPilot 안의 슬롯에 끼워져 사용된다. 사용자는 메모리의 확장을 위해 이 메모리 카드를 교체할 수 있다.
이렇게 많은 memory address space를 사용할 수 있긴 하지만 실제 사용하는 메모리 용량을 감안하여 Palm OS[2]는 작은 용량의 RAM에서 보다 효율적으로 동작할 수 있도록 설계되어 있다. 32K의 RAM을 stack, global, temporal memory allocation에 사용할 수 있도록 할당하고, 나머지 부분에 user

[2] PalmPilot의 운영체제로서 ROM에 저장된다.

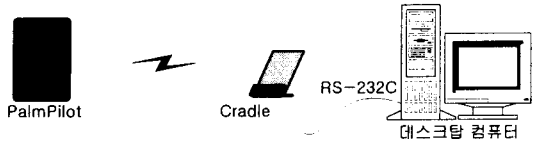

그림 1 **PalmPilot** 과 데스크탑과의 연결 구성도

data를 저장할 수 있다. 이러한 분할 중에서 전자를 dynamic RAM, 후자를 storage RAM이라고 부른다.

Palm OS는 제한된 메모리 용량과 disk 대신 nonvolatile RAM을 사용하기 때문에 일반적인 화일 시스템과 비교할 때 다음과 같은 커다란 차이점을 보인다.

일반 화일 시스템은 화일 전체 혹은 일부를 디스크에서 메모리 버퍼로 읽어들인 다음 메모리 버퍼 안에서 화일에 대한 작업을 하고 수정된 부분은 다시 디스크로 저장한다. 따라서 대용량의 메모리 버퍼를 요구한다. 하지만 Palm OS는 화일에 해당하는 데이타를 메모리에 저장하기 때문에 이에 대한 처리는 데이타가 저장된 곳에서 바로 행해진다. 따라서 일반 화일 시스템처럼 메모리 버퍼가 따로 필요 없다.

또한 거의 대부분의 사용자 데이타는 256bytes 이하의 작은 단위로 나뉠 수 있기 때문에 이들을 관리하는 메모리 매니저만 있으면 된다.

2.1 Memory structure

위에서 보았듯이 Palm OS는 작은 데이타 조각들을 효율적으로 관리해야 하므로 메모리 공간은 64K 이하의 heap 단위로 쪼개서 관리한다. Heap 내부에는 실제로 데이타가 들어가는 chunk가 있게 된다.

앞에서 PalmPilot에는 데스크탑의 메인 메모리에 해당하는 dynamic RAM과 디스크에 해당하는 storage RAM이 있는 것을 봤다. 이들은 각각 dynamic heap과 storage heap으로 나뉘고 memory manager와 data manager가 관리하게 된다.

2.1.1 Heap overview

Heap은 64K 이하의 연속된 메모리 공간으로 그 안에는 실질적인 사용자 데이타인 memory chunk가 들어가고 heap이

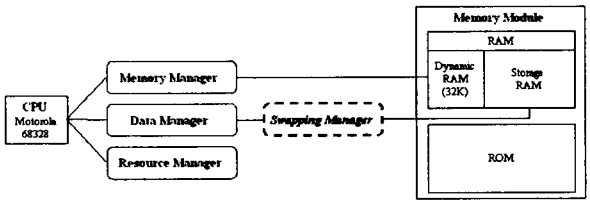

그림 2 **PalmPilot** 메모리 아키텍처

이들을 관리한다. 응용프로그램에서는 특정한 heap에 새로운 chunk를 만들어 사용할 수 있다. Memory manager는 각각의 heap을 개별적으로 관리하며 heap안의 chunk는 수시로 정렬할 수 있다.

Palm OS 환경에서 heap은 heap ID로 구분된다. Heap ID는 16-bit의 크기를 갖으며 전체 memory 영역에서 유일한 값을 갖는다. 맨 처음의 heap의 ID는 0이며 항상 dynamic heap으로 사용된다. Dynamic heap에는 일반 데스크탑에서의 메인 메모리처럼 응용프로그램이 사용하는 stack 등과 같은 run-time variable이 저장되며 응용프로그램이 종료되면 자신이 사용하던 dynamic heap을 반환한다.

Dynamic heap이외의 모든 영역은 일반 데스크탑에서의 디스크와 같은 역할을 하는 storage heap으로 사용된다.

2.1.2 Heap structure

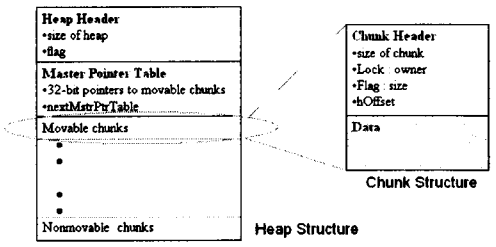

그림 3 **Heap** 과 **Chunk Structure**

Heap의 앞부분에는 그림 3과 같이 차례로 heap header, master pointer table (MPT), heap chunk가 위치한다.

· **Heap header** : heap의 맨 앞에 위치하며 heap의 크기와 memory manager가 참고할 정보들을 flag 형태로 저장한다.

· **Master pointer table** : heap 내의 chunk에 대한 32-bit 포인터의 table이다. Memory manager가 heap 내부의 chunk들을 compaction시 이들 포인터가 update 된다. 응용프로그램이 chunk를 사용할 때는 항상 이 MPT에서 사용할 chunk에 대한 handle을 얻은 뒤 사용하게 된다. 만일 MPT가 꽉 차게 되면 또 다른 MPT을 만들고 이전 MPT의 nextMstrPtrTable 필드에 링크 시킨다. 이런 식으로 계속 MPT을 만들어 나갈 수 있다.

· **Heap chunks** : 실제 데이타가 저장되는 부분이다. Chunk는 다시 movable chunk와 nonmovable chunk로 나뉘는데 ROM에 있는 모든 chunk들은 nonmovable chunk로서 MPT에는 이들 chunk에 해당하는 entry가 없다. Movable chunk는 heap에서 MPT이후 맨 앞에서부터, nonmovable chunk는 heap의 맨 뒤에서부터 allocate된다.

2.1.3 Chunk structure

Chunk는 메모리에서 실제 데이타를 저장하는 영역으로 그림 에서 보듯이 앞부분에는 여러가지 정보를 포함하고 있다.

· **Chunk header** : header는 6-byte로, chunk의 크기가 기록된다.
· **Lock:owner byte** : 1-byte 크기며, 상위 4-bit에는 lock count, 하위 4-bit에는 owner ID 가 기록된다. Lock count는 chunk가 lock될 때마다 1씩 증가하고, unlock될 때마다 1씩 감소한다. Movable chunk는 최대 14까지 증가할 수 있으며 nonmovable chunk는 항상 15의 값을 갖는다. Owner ID는 새로운 chunk가 만들어질 때마다 memory manager에 의해 기록되며 응용프로그램이 비정상적으로 종료된 후 debugging과 garbage collection에 유용하게 사용된다.
· **Flags:size adjustment byte** : 1-byte 크기며, 상위 bit에는 flag가 할당되어 있고 현재는 1 bit만 사용되며 free chunk일 경우 set 된다. 하위 bit에는 실제 data의 크기가 저장된다. 이는 chunk header로부터 얻어진 chunk의 크기에서 header 등의 크기를 뺀 값이 저장된다. 이 크기는 항상 2의 배수로서 항상 word단위로 시작될 수 있게 한다.
· **hOffset word** : 1-word의 크기로 MPT에서 해당 chunk의 entry로부터 chunk header까지의 거리를 반으로 나눈 값이다. 만일 MPT가 chunk보다 상위 address에 존재할 경우 이 값은 음수가 될 수도 있으며, nonmovable chunk의 경우에는 MTP에 entry가 없으므로 이 값은 0이 된다.

2.1.4 Memory manager

응용프로그램이 dynamic heap을 사용하기 위해서는 memory manager를 사용한다. Memory manager가 제공하는 주요 API는 다음과 같다.

· **MemHandleNew** : 새로운 movable chunk를 할당하기 위해 사용되며 원하는 크기를 파라메터로 넘긴다. 리턴 값은 성공했을 경우는 새로 만들어진 chunk에 대한 handle이며 실패할 경우 0이 리턴된다.
· **MemHandleLock** : chunk를 읽거나 쓰기 전에는 항상 lock을 걸어야 한다. Lock을 걸때마다 chunk의 LOCK 값이 하나씩 증가한다. 리턴 값은 해당 chunk에 대한 pointer가 된다.
· **MemHandleUnlock** : Lock을 걸고 사용하던 chunk를 unlock 시킬 때 사용한다.
· **MemHandleSize** : 어떤 chunk의 size를 알고 싶을 때 사용한다.
· **MemHandleResize** : 어떤 chunk의 size를 조정할 때 사용한다.
· **MemHandleFree** : 사용하던 chunk를 release 시킬 때 사용한다.

2.2 Storage structure

Palm OS에서는 메모리의 일부분을 기존의 화일 시스템의 디스크처럼 사용한다. Palm OS는 기존의 화일 시스템이 화일을 디스크로부터 메모리 버퍼로 읽어들인 뒤 메모리 상에서 작업한 내용을 다시 디스크 화일로 저장하는 것과 달리 모든 내용을 메모리에 저장하고 이에 대한 작업은 데이타가 저장된 메모리에서 바로 이루어 진다. 이러한 메모리 내의 사용자 데이타에 대한 관리를 data manager가 담당한다. 우리가 제안하는 swapping 메커니즘은 이 data manager의 기능을 일부 확장 시켜서 모든 응용프로그램이 사용할 확률이 가장 작은 사용자 데이타의 일부를 데스크탑으로 이동시키고 그 빈 공간만큼의 영역에 다시 새로운 데이타를 저장함으로써 보다 많은 데이타를 사용자로 하여금 사용할 수 있게 하는데 목표를 두고 있다.
따라서 기존의 data manager의 메커니즘을 살펴보고 이를 어떻게 확장 시키는지에 대해서 논해보고자 한다.

2.2.1 Database structure

Data manager가 처리하는 데이타는 기존 화일 시스템의 디스크 화일에 해당한다. Palm OS에서는 이를 *databases*라 부르고 이를 더 잘게 쪼갠 단위를 *record*라고 한다.
PalmPilot의 주요 응용분야가 주소록, 일정관리 등임을 감안할 때, record는 주소록이나 일정관리 프로그램에서 각각의 record와 일치한다.
Database와 record는 앞서 말한 memory structure위에서 구현된 것이기 때문에 database는 heap에 record는 heap안의 chunk로 구현된다. 또한 data manager는 memory manager가 제공하는 하위 모듈을 그대로 사용한다.
그림 Database header
Database도 heap과 마찬가지로 database header를 갖는다. 이는 heap 구조에서 heap header를 대체한다.
Database header는 그림 4와 같이 여러가지 필드를 갖는다.

· **name** : database의 이름을 기록한다.
· **attribute** : database의 속성에 대한 flag로써 사용된다.
· **version** : 응용프로그램별로 각각 사용되는 버젼 정보가 기록된다.
· **modificationNumber** : database에서 record가 삭제, 추가, 혹은 변경될 때마다 증가된다. 이는 데스크탑과의 hot synch 시 time stamp의 역할을 한다.
· **appInfoID** : 응용프로그램이 사용할 수 있는 optional 필

드다.

· **sortInfoID** : 위와 마찬가지로 응용프로그램이 사용할 수 있는 optional 필드다. Database에 대한 sort table에 대한 정보를 저장할 수 있다.

· **type과 creator** : Database들이 속한 응용프로그램을 구분하기 위해 사용된다. 기본적으로는 한 응용프로그램이 사용하는 database는 다른 응용프로그램에서 사용할 수 없다.

· **numRecords** : Database header안에 속해있는 record entry의 개수를 기록한다.

· **nextRecordList** : Database header안에 더 이상 record entry가 들어갈 수 없을 때 다른 곳에 record list를 만들고 이곳에 링크 시킨다.

· **record entry** : 8-byte 크기로 local ID, attribute, unique ID의 세 가지 필드로 구분된다.

2.2.2 Database header에서의 record entry structure

그림 에서처럼 database header에서 record entry는 각각 4 byte의 local ID, 8개의 attribute bits, 3 byte의 unique ID, 세 가지의 필드를 갖는다.

· **Local ID** : 메모리 카드 내에서의 database의 ID로서 이 ID를 사용하여 data manager는 해당 database의 handle을 리턴한다.

· **unique ID** : 전체 메모리 영역에서 유일한 database의 ID로서 데스크탑과의 백업과 동기화에 사용된다.

· **delete bit** : 사용자가 해당 record를 지울 때 set 되며 이 때 Local ID도 지워진다. 백업의 경우에는 delete bit만 set하고 Local ID는 그대로 남겨둠으로써 나중에 Cradle과 연결됐을 때 구분할 수 있도록 한다.

· **dirty bit** : 해당 record에 수정이 가해지면 set 된다. 이는 나중에 데스크탑과 동기화를 시킬 때 사용된다.

· **busy bit** : 응용프로그램이 해당 record를 read 또는 write하기 위해 lock을 걸었을 때 set 된다.

· **secret bit** : 이 bit가 set 되어 있다면 해당 record를 억세스하기 위해 패스워드가 필요하다.

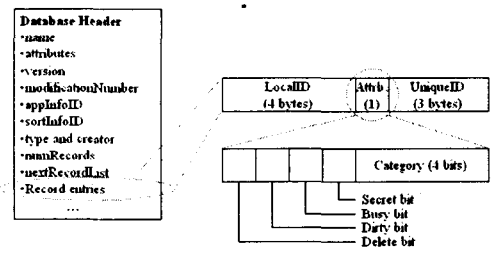

그림 **4 database header**

· **category** : 사용자는 모든 데이타를 최대 16가지로 임의로 분류할 수 있다. 예를 들어 "personal", "business", "course work" 등으로 분류할 수 있다.

2.2.3 Data manager

Data manager는 일반 file manager와 비슷한 역할을 한다.
앞서 말했듯이 화일에 해당하는 database는 여러개의 record로 나뉘어지고, 이들은 각각 메모리 heap과 chunk에 대응된다.
Data manager도 database와 record를 관리하기 위한 API들을 제공한다. 그리고 이들 API의 하부 구조에서는 memory manager의 유사한 기능을 하는 API를 사용한다.

· **DmCreateDatabase** : 새로운 database를 만든다.

· **DmDeleteDatabase** : 지정한 database를 삭제한다.

· **DmFindDatabase** : 메모리 영역에서 database의 이름으로 찾은 database의 Local ID를 리턴 한다. Database에 대한 모든 작업은 database의 Local ID를 필요로 하다.

· **DmGetDatabase** : 메모리카드의 특정 index에 위치한 database의 Local ID를 리턴한다.

· **DmGetRecord** : record index를 파라메터로 받아서 해당 record를 busy 상태로 마크한 뒤 record에 대한 handle을 리턴한다. 만일 이미 busy 상태였다면 에러를 리턴한다.

· **DmQueryRecord** : 응용프로그램이 특정 record의 데이타를 단지 읽기만 원한다면 busy 상태와는 상관 없이 사용할 수 있다. 이 경우 사용되며 다음의 DmReleaseRecord를 부를 필요가 없다.

· **DmReleaseRecord** : DmGetRecord를 한 뒤 사용이 끝난뒤 호출해야 한다. Busy bit을 reset하고 내용이 변경됐을 경우 dirty bit을 set한다.

· **DmRemoveRecord** : database header에서 해당 record entry를 삭제한다.

· **DmDeleteRecord** : database header에서 해당 record entry를 삭제하지 않고 delete bit을 set하고 Local ID를 지운다.

· **DmArchiveRecord** : 단지 delete bit만 set 한다. 따라서 이 record는 다음에 Cradle에 연결되면 데스크탑에 백업된 뒤 지워진다.

3. Swapping Manager

이제부터는 본 연구에서 제안하는 스와핑 매니져와 이의 메커니즘에 대해 설명하도록 한다. 기존의 PalmPilot에서는 그림 에서 보듯이 데스크탑과 Cradle을 이용해 시리얼 포트에 연결한 뒤 메모리에 저장된 데이타를 백업(archive)하고 동기화 시키기도 한다.

동기화란, 데스크탑에서도 PalmPilot의 어떤 응용프로그램과 같은 기능을 하는 프로그램에서 동일한 데이타에 대해 수정을 할 경우 이 둘을 가장 최신 버젼으로 맞추는 것을 말한다.
백업은 사용자가 임의로 선택한 데이타를 데스크탑에 저장시키고 PalmPilot의 메모리 영역에서는 삭제 하는 것을 말한다.
우리가 제안하는 스와핑 메커니즘은 자동화된 백업이라고도 말할 수 있다. 백업과 여기서 제안하는 스와핑과의 차이점은 사용자가 대상을 임의로 지정하지 않고 시기 또한 OS가 필요할 때 수행하는 점이 다르다.
메모리의 사용량이 늘어 빈 공간이 충분치 않을 때 OS는 사용자에게 Cradle에 연결할 것을 권유하고 연결이 된 뒤에는 swap-out 시킬 대상을 선정하여 데스크탑에 저장하고 PalmPilot의 메모리 영역에서는 삭제한다.
이 때 사용자는 어떤 메모리 영역이 swap-out 됐는지는 알지 못한다. 그 뒤 사용자가 swap-out된 데이타를 사용하고자 할 때 OS는 fault가 일어났음을 알리고 계속 작업을 진행하기 위해서는 다시 Cradle에 연결할 것을 권한다.
다시 데스크탑과 연결되면 OS는 앞에서 fault가 일어난 data를 다시 PalmPilot의 메모리로 로드한다.
여기서 중요한 세 가지는 스와핑이 일어나는 시점과 대상 선정, fault가 일어난 뒤의 처리 방법이다. 앞으로는 이들 각각에 대해 자세히 설명한다.

3.1 스와핑 시점

우리는 스와핑이 일어나는 시점을 메모리 사용량이 정해진 threshold를 넘어설 경우로 정했다. 이는 다시 두 단계로 나뉘는데 첫 단계는 빈 공간이 전체의 10% 이하로 떨어질 때 OS는 사용자에게 Cradle에 연결할 것을 권유한다. 하지만 사용자는 계속 작업을 수행할 수 있다.
다음 빈 공간이 전체의 5% 이하로 떨어질 경우 OS는 더 이상의 메모리 할당을 중단한다. 이후에는 반드시 Cradle에 연결 해야만 한다.

3.2 스와핑 대상 선정

앞에서 OS의 권유에 따라 사용자가 PalmPilot을 Cradle에 연결했을 경우 OS는 swap-out시킬 record를 선정한다. 여기서는 간단한 LRU 알고리즘을 사용한다. 이의 구현에는 stack을 사용한다. 앞에서 database header에는 record entry가 있음을 봤다. 이 record entry list를 stack 형태로 구현하여 reference 된 record의 entry는 항상 stack의 top으로 이동시킨다. 이렇게 하면 swap-out 시킬 victim은 stack의 bottom에 있는 record가 된다.

3.3 오류 처리

Swap manager는 사용자가 swap-out 된 record를 사용하고자 하면 에러 메시지를 표시하고 더 이상 진행하지 않는다. 이와 함께 swap manager의 fault record list에 이 record를 추가시킨다.
여기서 fault가 생겼을 때 더 이상 진행할 수 없는 것은 PalmPilot과 데스크탑과의 통신이 두절된 상태이기 때문에 불가피하다. 이런 처리 방법은 CODA 화일 시스템 [4] 에서도 적용하고 있다. CODA의 disconnected operation 상태에서의 cache miss에서도 이와 같은 방법을 제시하고 있는 것이다.

3.4 스와핑

이제 사용자가 PalmPilot을 Cradle에 연결했을 때, 어떤 식으로 스와핑이 일어나는지 알아보자.
우선 OS는 절에서 살펴본 victim 선정 방법으로 swap-out 시킬 record를 골라서 데스크탑에 저장 시킨다.
여기서 일반적인 백업과의 차이점은 백업의 경우 우선 database header에서 사용자가 선택한 record의 entry의 delete bit을 set한다. 이 때 Local ID는 그대로 남겨둠으로써 진짜 지운 record와 구분한다.
백업이 끝났으면 data manager는 database header에서 백업된 record의 entryf를 삭제하고 MPT에서도 역시 해당 record의 entry를 지운다. 이 점이 스와핑과 백업과의 차이점이 된다.
즉, swap-out의 경우 database header에서 record의 entry는 그대로 남겨두고 MPT에도 entry를 삭제하는 대신 pointer 값을 NULL로 바꾼다. 따라서 나중에 swap-out된 record를 구분할 수 있게 한다.
Record를 swap-out 시킬 때, 해당 record가 속해있던 database name과 record의 Local ID, 실제 데이타를 데스크탑에 저장한다.
이런 식으로 빈 메모리 공간을 확보한 뒤 swap manager에 기록된 fault record list에 들어있는 record들을 swap-in 시킨다.

4. 앞으로 할 일

우리는 본 연구에서 PalmPilot의 OS에서 데스크탑을 이용한 스와핑 방법을 고안했다. 하지만 이 방법은 단순히 이론적인 방법일 뿐 실제로 구현했을 때 제대로 동작하는지는 실험해보지 못했다. 그 이유 중 가장 큰 원인은 Palm OS의 소스를 구할 수 없기 때문에 위의 내용을 추가시킬 수 없다.
가장 좋은 방법은 보다 체계적이고 완벽한 연구를 한 뒤 USRobotics 사와 접촉하여 실제 Palm OS 상에서 구현해보고 효용성이 입증 된다면 차기 Palm OS에 실제로 탑재하는 것이다.

5. 결 론

우리는 PalmPilot PDA에서의 가장 큰 문제점 중 하나인 메모리의 제약을 해결하기 위해 Disconnected swapping 기법을 제안하였다. 사용자는 더 이상 메모리 용량에 신경을 쓰지 않고 마음대로 사용해도 된다.
아쉬운 점이 있다면 fault가 일어났을 때 응용프로그램은 해당 record에 대한 작업을 더 이상 진행할 수 없다는 점이다. 하지만 이는 PalmPilot 자체의 communication 기능이 없기 때문으로 현재로서는 어떻게 해결할 수 없는 문제라고 생각된다.
또 한가지 실제 Palm OS 상에서 구현해보고 성능 평가를 해볼 수 없었다는 점이 남는다.

참고문헌

[1] The Pilot Page , http://www.pilot.org/

[2] PalmPilot Software, News, Links, and More ,

[3] PalmPilot Development Resources at ReadCoders , http://www.roadcoders.com/pilot/

[4] Satyanarayanan, M., Kistler, J.J., Mummert, L.B., Ebling, M.R., Kumar, P., Lu, Q.
Experience with Disconnected Operation in a Mobile Computing Environment.
In *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing.* Cambridge, MA, August, 1993

[5] Developing Palm OS Applications Part I , http://palmpilot.3com.com/5024.html

[6] Developing Palm OS Applications Part II , http://palmpilot.3com.com/5024.html