System Operacyjny
“Alt Ctrl Delete”
Specyfikacja

x86

Version 0.01

Karol Gasiński
kuktus@gmail.com
www.mrkaktus.org

31 maja 2007

Spis treści

1 Architektura Systemu

1.1 Wstęp

System operacyjny Alt Ctrl Delete (w skrócie ACD) jest eksperymentalnym projektem mającym na celu przeprowadzenie badań nad niskopoziomowym zarządzaniem zasobami sprzętu komputerowego. Ze względu na przeniesienie większości funkcjonalności do hostów poziomu użytkownika, system jest klasyfikowany jako µkernel (mikrokernel). Całość kodu jest kompilowana za pomocą NASM'a.

1.1.1 Główne założenia

Szybkość działania — System projektowany jest z myślą o jak największej wydajności, dlatego też kernel jest w całości napisany w Asemblerze, a każda funkcja jest wielokrotnie przepisywana dla osiągnięcia maksymalnej optymalizacji.

Uniwersalność — Cała architektura systemu została oparta o mikrokernel w celu uzyskania systemu składającego się z dużej liczby współpracujących ze sobą modułów, które można konfigurować według własnych potrzeb. Dzięki takiej budowie, system można dowolnie dostosowywać otrzymując maksymalną wydajność w różnych zastosowaniach.

Bezpieczeństwo — W celu zapewnienia należytej ochrony dla procesów system wykorzystuje udostępnione w architekturze x86/IA32 mechanizmy sprzętowe takie jak Tryb Chroniony czy stronicowanie pamięci.

1.1.2 Minimalne wymagania

Dzięki swej zwięzłej i zoptymalizowanej budowie, system ma bardzo małe wymagania sprzętowe, które w dzisiejszych czasach spełnia każdy komputer:

- Procesor 80386-SX 16MHz lub lepszy (niewymagany koprocesor).
- Pamięć operacyjna 1MB RAM.
- Karta graficzna kompatybilna z VGA.
- Napęd dyskietek 5.25" lub 3.5" lub inny napęd dysków wymiennych.
- Klawiatura.
- Monitor.

1.1.3 Ograniczenia

Na dzień dzisiejszy system posiada pewne ograniczenia, które w przyszłości zostaną usunięte:

- Maksymalnie obsługiwane 4GB pamięci RAM.
- Kod procesu może zajmować nie więcej niż 4MB.
- Pamięć dostępna dla procesu to maksymalnie (4GB – 4MB) wraz ze stosem.
- Nieobsługiwana jest wielowątkowość ani wieloprocesorowość.

1.2 Start Systemu

Ze względu na jądro typu mikrokernel, system ACD jest uruchamiany w specyficzny sposób. Kod ładujący kernel dodatkowo ładuje do pamięci aplikacje startowe systemu, określone wcześniej przy konfiguracji dysku startowego. Właścicielem tych aplikacji jest root, a rodzicem system. Taki sposób startu systemu umożliwia pełną elastyczność przy jego konfiguracji. Przykładowo można uruchomić razem z jądrem pojedynczy proces wykonujący specjalistyczne zlecone obliczenia, do których potrzebuje on pełnej mocy obliczeniowej komputera, nie potrzebuje zaś np. serwera klawiatury czy sieci.

1.2.1 Etap 1 – Bootloader

Załadowany i uruchomiony przez BIOS, bootloader ze względu na swój bardzo mały rozmiar, ma za zadanie załadować i uruchomić bootloader drugiego stopnia (dalej nazywany również SSBL – *Second Stage Bootloader*). SSBL nie jest już ograniczony, jeżeli chodzi o rozmiar i może zawierać wszystkie niezbędne procedury.

Aby załadować drugi stopień, bootloader musi użyć systemu plików znajdującego się na nośniku, z którego został załadowany. Do tego celu bootloader może załadować do pamięci struktury zarządzania danym FS'em, które później są pozostawione do użytku drugiego stopnia. Wszystkie tymczasowe dane jak dodatkowe struktury czy sam drugi stopień, są ładowane poniżej 639 kilobajta pamięci zaczynając od najwyższych adresów. Po skonfigurowaniu swojego środowiska bootloader ładuje z pliku SSBL i przekazuje mu kontrolę. W przypadku nie odnalezienia pliku SSBL, zostaje wyświetlony znak błędu i komputer zostaje zawieszony do czasu ponownego restartu.

Adres początkowy	Rozmiar	Opis
00000 – 0KB – 000000	1KB	Tablica wskazów przerwań RM
00400 – 1KB – 001024	172B	BDA (Bios Data Area)
004AC – 1196B – 001196	68B	Zarezerwowane przez IBM
004F0 – 1264B – 001264	16B	Obszar komunikacji użytkownika
00500 – 1280B – 001280	2816B	Obszar danych (A)
01000 – 4KB – 004096	12KB	Obszar zmiany kontekstu. (B)
04000 – 16KB – 016384	15KB	———
07C00 – 31KB – 031744	0.5KB	Bootloader 1.
07E00 – 31.5KB – 032256	588.5KB	———
9B000 – 620KB – 634880	3.5KB	Bootloader 2. (C)
9BE00 – 623.5KB – 638464	4.5KB	Bootloader – FAT
9D000 – 628KB – 643072	4KB	Bootloader – Bufor odczytu
9E000 – 632KB – 647168	7KB	Bootloader – Root Dir
9FC00 – 639KB – 654366	1KB	EBDA (Extended BIOS Data Area)
A0000 – 640KB – 655360	64KB	Pierwszy ekran k.graficznej
B0000 – 704KB – 720896	32KB	Drugi ekran k.graficznej (jego 1 połowa) LUB 8 planów ekranu tekstowego (MONO)
B8000 – 736KB – 753664	32KB	Drugi ekran k.graficznej (jego 2 polowa) LUB 8 planów ekranu tekstowego (COLOR)
C0000 – 768KB – 786432	32KB	BIOS Karty Graficznej
C8000 – 800KB – 819200	160KB	BIOS Shadow Area
F0000 – 960KB – 983040	64KB	BIOS

Tab. Stan pierwszego megabajtu pamięci RAM po zakończeniu działania Bootloadera.

1.2.2 Etap 2 – Bootloader drugiego stopnia

Zadaniem bootloadera drugiego stopnia jest wczytanie do pamięci i uruchomienie głównego programu ładującego system (zwanego dalej SL – *System Loader*) i pośredniczenie pomiędzy nim a nośnikiem, z którego system startuje. Główną udostępnianą funkcją jest możliwość odczytania dowolnego pliku pod dowolny adres pamięci. Aby to umożliwić drugi stopień przełącza procesor w Unreal Mode (Real Mode z adresacją pełnej szyny adresowej). Dzięki zastosowaniu takiego podziału SL jest uniwersalny dla wszystkich nośników danych używanych do startu systemu. Bootloader drugiego stopnia pozostaje w pamięci przez cały czas pracy SL wykonując jego zadania obsługi nośnika.

Adres początkowy	Rozmiar	Opis
00000 - 0KB - 000000	1KB	Tablica wskazów przerwań RM
00400 - 1KB - 001024	172B	BDA (Bios Data Area)
004AC - 1196B - 001196	68B	Zarezerwowane przez IBM
004F0 - 1264B - 001264	16B	Obszar komunikacji użytkownika (?)
00500 - 1280B - 001280	2816B	Obszar danych (A)
01000 - 4KB - 004096	12KB	Obszar zmiany kontekstu. (B)
04000 - 16KB - 016384	15KB	———
07C00 - 31KB - 031744	0.5KB	Bootloader 1.
07E00 - 31.5KB - 032256	584.5KB	———
9A000 - 616KB - 630784	4KB	System Loader
9B000 - 620KB - 634880	3.5KB	Bootloader 2. (C)
9BE00 - 623.5KB - 638464	4.5KB	Bootloader - FAT
9D000 - 628KB - 643072	4KB	Bootloader - Bufor odczytu
9E000 - 632KB - 647168	7KB	Bootloader - Root Dir
9FC00 - 639KB - 654366	1KB	EBDA (Extended BIOS Data Area)
A0000 - 640KB - 655360	64KB	Pierwszy ekran k.graficznej
B0000 - 704KB - 720896	32KB	Drugi ekran k.graficznej (jego 1 połowa) LUB 8 planów ekranu tekstowego (MONO)
B8000 - 736KB - 753664	32KB	Drugi ekran k.graficznej (jego 2 polowa) LUB 8 planów ekranu tekstowego (COLOR)
C0000 - 768KB - 786432	32KB	BIOS Karty Graficznej
C8000 - 800KB - 819200	160KB	BIOS Shadow Area
F0000 - 960KB - 983040	64KB	BIOS

Tab. Stan pierwszego megabajtu pamięci RAM po zakończeniu działania SSBL.

1.2.3 Etap 3 – Program ładujący

Po przekazaniu kontroli do programu ładującego rozpoczyna on swoją pracę od załadowania jądra systemu. Następnie inicjuje on struktury podstawowego Menedżera Pamięci RAM niezbędnego do prawidłowego załadowania aplikacji root'a. Struktury te będą później wykorzystywane przez kernel. Gdy MM jest gotowy, program ładujący wczytuje do pamięci plik INIT.DAT przechowujący informacje o aplikacjach root'a które należy uruchomić. Kolejny krok to załadowanie aplikacji do pamięci, (przy czym aplikacje są tylko ładowane, nie występuje żadna ich inicjacja). Adresy, pod które załadowano programy są zapisywane w tablicy inicjacji procesów (odczytanej z INIT.DAT) dla przyszłego użytku jądra. Gdy wszystkie te kroki zostaną wykonane program ładujący przełącza procesor w Protected Mode i przekazuje kontrole kernelowi.

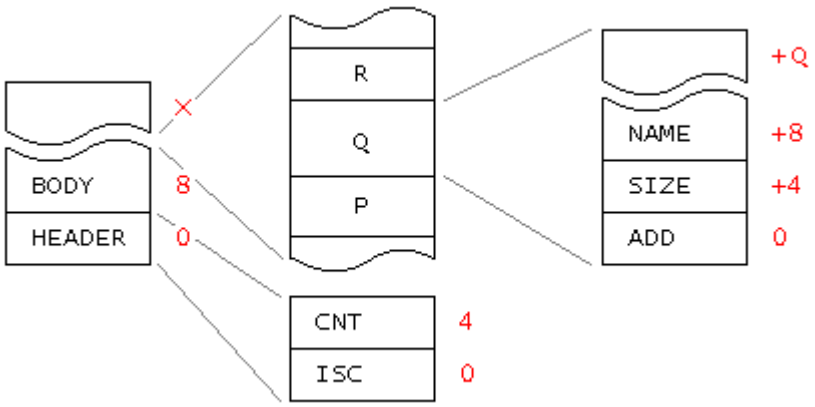
Adres początkowy	Rozmiar	Opis
00000 - 0KB - 000000	1KB	Tablica wskazów przerwań RM
00400 - 1KB - 001024	172B	BDA (Bios Data Area)
004AC - 1196B - 001196	68B	Zarezerwowane przez IBM
004F0 - 1264B - 001264	16B	Obszar komunikacji użytkownika
00500 - 1280B - 001280	2816B	Obszar danych (A)
01000 - 4KB - 004096	12KB	Obszar zmiany kontekstu. (B)
04000 - 16KB - 016384	16KB	Kernel
08000 - 32KB - 032768	4KB	GTB
09000 - 36KB - 036864	4KB	TB (4KB dla do 128MB RAM)
0A000 - 40KB - 040960	4KB	ETB (4KB dla do 128MB RAM)
0B000 - 44KB - 045056	X	APLIKACJE
	572 -X -Y	---
	Y	PLIK INICJACJI APLIKACJI
9A000 - 616KB - 630784	8KB	TEMP
9C000 - 624KB - 638976	8KB	Kernel - IOPRBM
9E000 - 632KB - 647168	4KB	Kernel - Stos
9F000 - 636KB - 651264	3KB	R
9FC00 - 639KB - 654366	1KB	EBDA (Extended BIOS Data Area)
A0000 - 640KB - 655360	64KB	Pierwszy ekran k.graficznej
B0000 - 704KB - 720896	32KB	Drugi ekran k.graficznej (jego 1 połowa) LUB 8 planów ekranu tekstowego (MONO)
B8000 - 736KB - 753664	32KB	Drugi ekran k.graficznej (jego 2 polowa) LUB 8 planów ekranu tekstowego (COLOR)
C0000 - 768KB - 786432	32KB	BIOS Karty Graficznej
C8000 - 800KB - 819200	160KB	BIOS Shadow Area
F0000 - 960KB - 983040	64KB	BIOS Systemu (glowny ladowany z ROMu)

Tab. Stan pierwszego megabajtu pamięci RAM w momencie uruchomienia kernela.

1.2.4 Tablica Inicjacji Procesów

Plik INIT.DAT zawiera strukturę zwaną Tablicą Inicjacji Procesów. Tablica ta zawiera wszelkie niezbędne dla programu ładującego informacje o aplikacjach root'a. Cały plik można podzielić na dwie części: nagłówek i ciało tablicy. Nagłówek tablicy składa się z dwóch pól:

ISC Rozmiar całego pliku INIT.DAT, pole to służy, jako suma kontrolna zabezpieczająca system przed użyciem uszkodzonego pliku INIT.DAT.
CNT Pole to przechowuje całkowitą ilość aplikacji do uruchomienia.



Rys. Struktura Tablicy Inicjacji Procesów pliku INIT.DAT.

Ciało tablicy składa się natomiast z CNT pól o zmiennym rozmiarze, z czego każde pole opisuje jedną aplikację (na rysunku wyszczególnione, jako pole „Q"):

ADD Pole to jest zawsze wyzerowane w pliku INIT.DAT, natomiast po załadowaniu aplikacji do pamięci przechowuje ono jej adres w RAM.
SIZE Jest to rozmiar aplikacji, jaki powinna ona posiadać. To pole jest zarówno przydatne przy uchronieniu systemu od załadowania uszkodzonej aplikacji jak i przy późniejszej inicjacji aplikacji przez kernel.
NAME Jest to ciąg znaków o zmiennej długości zakończony bajtem zerowym. Przechowuje on nazwę aplikacji.

1.2.5 Konfiguracja aplikacji startowych

W celu skonfigurowania własnej listy aplikacji do uruchomienia wraz z kernelem należy posłużyć się programem administracyjnym MAKEINIT. Utworzenie własnego pliku INIT.DAT za jego pomocą jest bardzo proste. Wystarczy umieścić program MAKEINIT w folderze, w którym przechowujemy aplikacje a następnie uruchomić go w wierszu poleceń z nazwami tych aplikacji, jako parametry jego wywołania. Program utworzy automatycznie plik INIT.DAT, który wraz z wcześniej wskazanymi aplikacjami należy przekopiować do głównego katalogu nośnika, z którego ma być uruchamiany system (np. w przypadku dyskietki FDD do folderu a:\). Jeżeli MAKEINIT zostanie wywołany w folderze z istniejącym już plikiem INIT.DAT plik ten zostanie nadpisany nowym.

1.3 Pamięć

System ACD w wersji 0.01 posiada prosty mechanizm zarządzania pamięcią. Ze względu na użycie mechanizmu stronicowania do zarządzania przestrzenią procesów, menedżer pamięci jądra posiada granularność alokacji wynoszącą 4KB. Z tego powodu mechanizmem zarządzającym pamięcią operacyjną są Tablice Bitowe. Podstawowa tablica bitowa TB i rozszerzona tablica bitowa ETB razem przechowują informacje o stanie ramek pamięci. Do każdej 4KB ramki pamięci przyporządkowany jest jeden bit w TB i jeden bit w ETB. Położenie tego bitu w tablicach to numer ramki, którą bit opisuje i tak dla przykładu ramce 0 odpowiada bit 0 w bajcie 0, natomiast ramce 34 odpowiada bit 2 w bajcie 4. Pobrane bity z TB i ETB razem tworzą parę opisującą ramkę (stąd możliwe są 4 stany ramki).

GTB	TB	ETB	Opis
0	0	0	Wolna ramka gotowa do alokacji.
1	0	1	Wolna ramka ze starymi danymi, czeka na czyszczenie.
1	1	0	NIEALOKOWALNA RAMKA: - ramka używana przez sprzęt - nieistniejąca fizycznie ramka - ramka uszkodzonego obszaru RAM
1	1	1	Ramka zarezerwowana, zaalokowana przez proces

Tab. Zależności stanów bitowych w trzech tablicach.

Przy maksymalnej ilości pamięci, TB i ETB przechowują informacje o 2^{20} ramkach (ponad jeden milion). Przeszukiwanie tak dużej tablicy w celu znalezienia wolnej ramki do alokacji trwałoby zbyt długo (nawet testując naraz po 32 bity trzeba by wykonać przy pesymistycznym założeniu 32768 porównań). Aby temu zapobiec istnieje trzecia tablica bitowa zwana Globalną Tablicą Bitową (GTB). Jeden bit w GTB opisuje 32 ramki. Jeżeli choć jedna z tych ramek jest wolna bit przyjmuje wartość 0, w przeciwnym wypadku przyjmuje wartość 1. Dzięki GTB nawet przy 4GB pamięci operacyjnej można za jednym porównaniem pobieżnie sprawdzić 256 ramek, co w pesymistycznym wypadku daje 1024 testy w celu odnalezienia wolnej ramki.

1.4 Procesy

Każdy proces w systemie ACD jest reprezentowany przez PDB (*Process Description Block*). Blok ten zawiera wszystkie niezbędne informacje o procesie, jego stan, skrzynkę wiadomości (patrz rozdział 1.8) oraz jeżeli zachodzi taka potrzeba kopie IOPBM (patrz rozdział 1.6). Każdy PDB ma rozmiar 4KB (lub 12KB wraz z kopią IOPBM) ze względu na używane w jądrze mechanizmy zarządzania pamięcią (patrz rozdział 1.4).

1.4.1 VPS
(Virtual Process Space)

Dzięki zastosowaniu mechanizmu stronicowania każdy proces posiada własną przestrzeń adresową zwaną także VPS. Pierwsze 4MB tej przestrzeni zajmuje zmapowana do niej przestrzeń jądra (VSS – *Virtual System Space*). Od adresu 4MB następuje obszar kodu procesu, a po nim rozszerzalna sterta danych. Na samym końcu VPS od adresu 4GB w dół znajduje się stos procesu.

Wersja jądra 0.01 udostępnia procesowi tak duży obszar VPS dzięki zastosowaniu VSS mapującej pamięć w stosunku 1:1 (adres fizyczny = adres wirtualny). Przy każdorazowym przejściu do poziomu jądra następuje przełączenie przestrzeni stronicowania, dzięki czemu kernel ma dostęp do całej pamięci operacyjnej (ta metoda jest jednak wolna z powodu dwóch przełączeń stronicowania, dlatego też w wersji 0.02 zostanie zastąpiona przez prace jądra w pełnym VSS).

1.4.2 Stany procesu

Proces w systemie ACD może się znajdować w jednym z czterech stanów. Podstawowym stanem jest „Nowy". Proces występuje w tym stanie, gdy zostaje wywołana funkcja jego utworzenia. Gdy zostaną utworzone wszystkie struktury systemowe, proces jest gotowy do wykonania i przechodzi w stan „Gotowy". W tym stanie oczekuje on na wykonanie. Gdy nadejdzie odpowiedni moment system przydzieli procesowi procesor przełączając go w stan „Działający". Przełączanie pomiędzy kontekstami w systemie ACD zostało zaimplementowane w sposób programowy ponieważ okazał się on szybszy od stosowania segmentów opisu procesów i mechanizmu sprzętowego. Gdy wyczerpie się kwant czasu przeznaczony dla pracy tego procesu lub zostanie wywołane API systemowe, lub też wystąpi inne przerwanie proces zostanie zatrzymany i przeniesiony z powrotem do kolejki „Gotowy" (ponieważ system ACD używa multitaskingu z wywłaszczeniem – *preemptive multitasking*).

Rys. Czterostanowy model procesu.

Może także wystąpić sytuacja, w której proces będzie oczekiwał na określoną wiadomość, wtedy zostanie on przeniesiony do kolejki procesów oczekujących do momentu nadejścia wskazanej wiadomości.

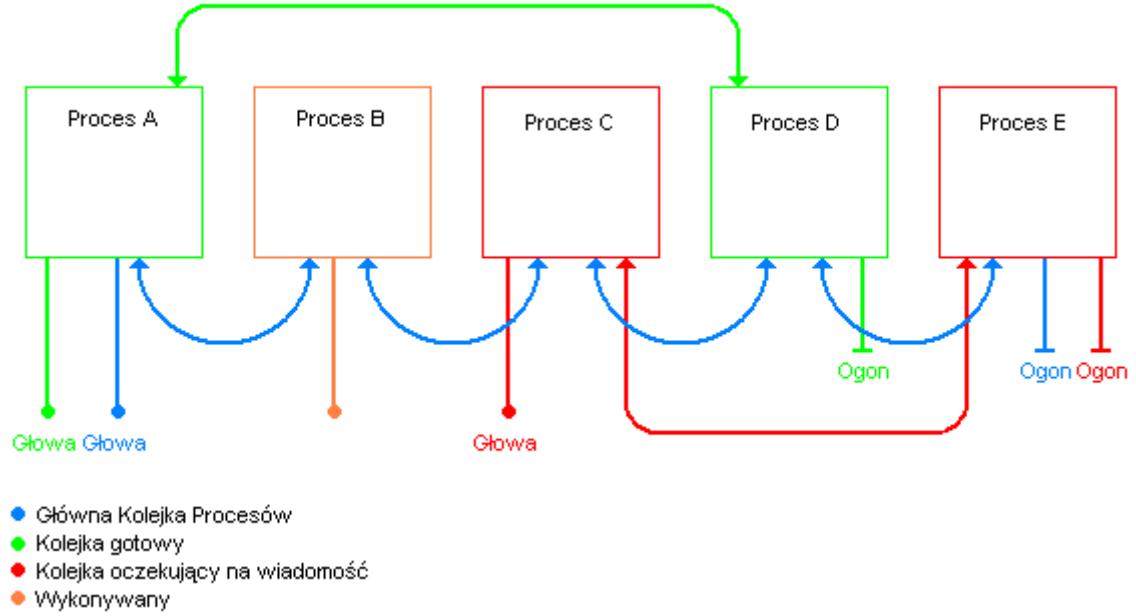
1.4.3 Process Description Block

Offset	Rozmiar	Nazwa	Opis
0	4B	PID	Numer identyfikacyjny procesu.
4	4B	FID	Numer identyfikacyjny procesu rodzicielskiego.
8	4B	UID	Numer identyfikacyjny właściciela.
12	4B	EAX	Stany rejestrów ogólnego przeznaczenia (#GP - General Purpose) podczas wystąpienia przerwania.
16	4B	EBX	
20	4B	ECX	
24	4B	EDX	
28	4B	ESI	
32	4B	EDI	
36	4B	EBP	
40	4B	EIP	Informacje odkładane przez procesor podczas wywołania przerwania i przywracane IRETD.
44	4B	EFLAGS	
48	4B	ESP	
52	4B	R	Zarezerwowane
56	4B	CR3	Adres Katalogu Stron (wskaz do VPS)
60	4B	IOPLBM1	Adres do Bitmapy dostępu portów I/O (część 1)
64	4B	IOPLBM2	Adres do Bitmapy dostępu portów I/O (część 2)
68	4B	GKPU	Poprzedni PDB w głównej kolejce procesów
72	4B	GKPD	Następny PDB w głównej kolejce procesów
76	4B	KSP	Poprzedni PDB w kolejce stanu
80	4B	KSN	Następny PDB w kolejce stanu
84	1B	STAN	Stan w jakim znajduje się aktualnie proces
85	3B	R	Zarezerwowane
88	1B	HV	Wysoka wersja programu
89	1B	LV	Niska wersja programu
90	2B	COMP	Numer kompilacji
92	1B	CBHV	Najstarsza wersja z która ten program jest w pełni zgodny wstecz.
93	1B	CBLV	
94	2B	CBCOMP	
96	32B	NAZW	Nazwa obrazu (32znakow, zero domyślne)
128	4B	CHSUM	Suma kontrolna dla adresów IPC Zawiera negacje adresu PDB
132	4B	MLBS	Głowa kolejki w skrzynce (IPC 2.0+)
136	4B	MLBE	Ogon kolejki w skrzynce (IPC 2.0+)
140	1B	MCNT	Ilość wiadomości w skrzynce (IPC 2.0+)
141	3B	R	Zarezerwowane
144	4B	MEMBOT	Dolna granica przestrzeni danych procesu
148	4B	MEMTOP	Górna granica przestrzeni danych procesu
152	4B	STABOT	Dolna granica stosu
156	4B	MEMFR	Ilość ramek zajmowanych przez proces (k+d+s)
160	1888B	R	Zarezerwowane
2048	2048B	MES	Skrzynka odbiorcza procesu, przechowuje do 128 wiadomości IPC, każda o rozmiarze 16 bajt (w tym 12 bajt na treść wiadomości i 4 nagłówka).

Tab. Budowa Struktury Opisu Procesu (PDB).

1.4.4 Mechanizm kolejkowania

Zarządzanie procesami odbywa się za pomocą dwukierunkowych kolejek z wskazami na głowę i ogon. Każdy proces należy do Głównej Kolejki Procesów, dzięki czemu podążając po niej można odczytać informacje o stanie wszystkich procesów w systemie. Druga kolejka, do której należy każdy proces to kolejka stanu. Kolejek stanu może być wiele jednak proces zawsze do jakiejś należy, (do której kolejki należy proces wskazuje pole STAN w jego PDB).



Rys. Przykład zależności kolejek i PDB.

Jak widać na przedstawionym powyżej diagramie, od strony jądra może istnieć wiele kolejek, jednak od strony PDB zawsze są to cztery wskazy opisujące dwie kolejki. W powyższym przykładzie Procesy A i D oczekują na wykonanie tworząc kolejkę „Gotowy”, procesy C i E tworzą kolejkę „Zawieszony” gdyż oczekują wiadomości. Proces B natomiast jest w stanie „Działający”.

1.5 Porty

W celu ochrony portów przed niepowołanym dostępem poszczególnych aplikacji, w systemie ACD zaimplementowano mechanizm zabezpieczający. Proces musi dokonać prośby o przydzielenie mu portu, jeżeli system zdecyduje, że można przydzielić dany port dokonuje tego i sygnalizuje to odpowiednim kodem powrotu z procedury. Jeżeli proces spróbuje dokonać odczytu lub zapisu na porcie, którego nie zarezerwował, system nie dopuści do tego i zakończy go. Po zakończeniu operacji na porcie proces powinien go zwolnić, aby inny proces mógł uzyskać do niego dostęp. Jeżeli proces nie zwolni portów zostaną one zwolnione dopiero przy jego zakończeniu. Mechanizm zarządzania portami pozwala na rezerwacje pojedynczych, podwójnych jak i poczwórnych portów. Wszystkie operacje związane z portami dostępne są poprzez grupę 3, przerwania 0x80.

1.5.1 Porty a architektura x86

Procesory rodziny x86 udostępniają sprzętowy mechanizm zarządzania procesami, ponieważ jest on jednak wolniejszy od mechanizmu programowego, w praktyce system ACD tworzy tylko jeden blok TSS (*Task State Segment*), który używany jest do opisu wszystkich procesów. Aby uzyskać ochronę portów blok TSS wskazuje na IOPBM (*I/O Permission Bit Map*). IOPBM opisuje każdy możliwy port za pomocą pojedynczego bitu, co łącznie daje jej rozmiar równy 8KB, (ponieważ w komputerze klasy PC może istnieć 2^{16} portów). Bit wyzerowany oznacza, że proces ma zezwolenie na używanie portu mu odpowiadającego, bit ustawiony oznacza, że takiego zezwolenia niema.

1.5.2 Implementacja ochrony portów

Ponieważ system ACD używa jednego bloku TSS dla wszystkich procesów, oznacza to, że zmiana wykonywanego procesu ponosi za sobą potrzebę uaktualnienia IOPBM. Problem ten został rozwiązany poprzez utworzenie prywatnych kopii tablicy IOPBM dla każdego procesu dokonującego żądania o port. Oznacza to, że proces pracujący na portach otrzymuje dodatkowy blok o rozmiarze 8KB przechowujący informacje o tym, jakie porty może on używać a jakich nie. Gdy proces taki jest wznawiany ochrona portów zostaje włączana w TSS a jego prywatna IOPBM zastępuje poprzednią IOPBM w TSS. Procesy niepracujące na portach nie posiadają własnych kopii IOPBM gdyż są one tworzone przy pierwszym żądaniu procesu o przydzielenie portu. Mechanizm ten zapobiega nadmiarowemu marnowaniu pamięci operacyjnej i przyspiesza przełączanie pomiędzy kontekstami procesów.

Podstawowym założeniem ochrony portów jest koncepcja prostego i zarazem bezpiecznego zarządzania zasobami. Aby tego dokonać przyjęto następującą zasadę: każdy port może być w danej chwili używany tylko przez jedną aplikację. Dzięki temu założeniu kernel wprowadza podstawowe zabezpieczenie pracy procesów na portach i co najważniejsze, umożliwia uruchomienie bardziej złożonej aplikacji użytkownika, która będzie ochraniać porty niezależnie od jądra.

Aby powyższa zasada mogła funkcjonować wprowadzono kolejny mechanizm, synchronizujący wszystkie kopie IOPBM. Jądro poza pojedynczym TSS z jego IOPBM tworzy IOPRBM (*I/O Protection Bit Map*). Każde żądanie i zwolnienie portów przechodzi przez mechanizm sprawdzający dany port w IOPRBM. Jeżeli bit portu w IOPRBM jest wyzerowany to oznacza to, że jeden z procesów już go zarezerwował, jeżeli bit jest ustawiony to port jest wolny do rezerwacji. W ten sposób IOPRBM jest jakby negacją złożenia wszystkich kopii IOPBM procesów.

1.6 Przerwania

System zarządzania przerwaniami nie jest zaimplementowany w tej wersji.

1.7 Komunikacja Międzyprocesowa

(IPC - Internal Process Communication)

Komunikacja między procesami jest udostępniana za pomocą grupy 5, przerwania 0x80. Ponieważ komunikaty IPC są podstawą modularnego systemu ich przekazywanie musi być ekspresowe. Z tego też względu zdecydowano się na następujące rozwiązania.

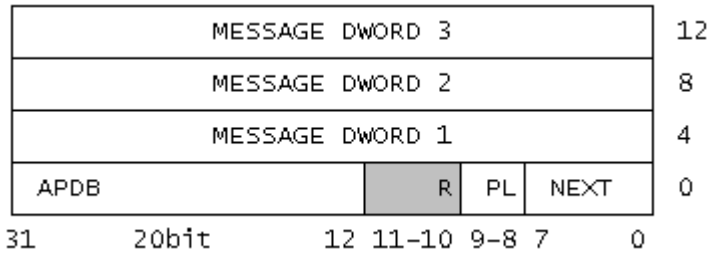
IPC w systemie ACD oparte jest na bezpośrednim przekazie wiadomości o stałym rozmiarze. Jest ono bardzo wydajne, ponieważ nie jest marnowany czas na wyszukiwanie wiadomości w wspólnych skrzynkach pocztowych. Każdy proces wraz z jego utworzeniem otrzymuje własną skrzynkę odbiorczą mogącą pomieścić do 128 wiadomości. Dodatkowo dla uzyskania maksymalnej przepustowości posłużono się adresowaniem wiadomości nie za pomocą identyfikatorów procesów, lecz adresów odbiorczych ich skrzynek (nowe mechanizmy jądra 0.02 zezwolą na adresowanie wiadomości identyfikatorami procesów).

1.7.1 Priorytety

Każda wiadomość może otrzymać priorytet ważności. Istnieją cztery poziomy uprzywilejowania, od zerowego (najniższego, domyślnego) oznaczającego zwykłą wiadomość, do trzeciego (najwyższego) oznaczającego wiadomość o najwyższym stopniu uprzywilejowania. Wiadomości z wyższymi priorytetami są dostarczane szybciej od tych z niższymi. Oznacza to, że wiadomość z poziomem uprzywilejowania 1 może zostać odebrana przed wiadomością dostarczoną do skrzynki wcześniej o priorytecie 0. Procesy posiadają pełną kontrolę nad tym, jaki priorytet chcą nadać swoim wiadomością.

1.7.2 Struktura wiadomości

Skrzynka odbiorcza każdego procesu ma stały rozmiar 2KB (zawiera 128 pól po 16 bajt). Każde pole składa się z 4 bajtów nagłówka i 12 bajt wiadomości.



Rys. Struktura wiadomości w skrzynce

APDB Adres PDB nadawcy.

R Pole zarezerwowane dla przyszłego użytku.

PL Poziom uprzywilejowania wiadomości. Może wynosić od 0 (zwykła wiadomość) do 3 (wiadomość o najwyższym priorytecie). Wiadomość o PL = 3 może zostać odebrana przed trzema wcześniej przyjętymi do skrzynki wiadomościami, wiadomość o PL = 2 przed dwoma itd.

NEXT Przechowuje numer pola, w którym znajduje się kolejna wiadomość. Pola są numerowane od 0 do 127.

1.7.3 Zarządzanie skrzynką

Wiadomości przychodzące są zorganizowane w postaci listy jednokierunkowej ze wskazami na jej początek (pole MLBS w PDB) i koniec (pole MLBE w PDB). Dodatkowo PDB zawiera pole MCNT przechowujące ilość znajdujących się w skrzynce wiadomości. Jeżeli skrzynka jest pusta, wszystkie trzy pola są wyzerowane, jeżeli jest pełna, pole MCNT jest równe 128. Zapis nowej wiadomości polega na liniowym przeszukaniu skrzynki

w celu odnalezienia wolnego pola, zapisaniu wiadomości i inkrementacji MCNT. Odczyt wiadomości powoduje przekopiowanie wiadomości do rejestrów procesu, odłączenie jej z kolejki i wyzerowanie pola, które zajmowała. Na końcu licznik MCNT jest dekrementowany.

2 Praca z Systemem

2.1 Kompilacja Programów

Ze względu na modularną budowę systemu, możliwe jest uruchomienie prawie dowolnego formatu pliku wykonywalnego. Dzieje się tak dzięki dwustopniowemu uruchamianiu aplikacji. Pierwszy stopień polega na wczytaniu do pamięci pliku programu wykonywalnego przez dowolną aplikacje poziomu użytkownika. Aplikacja ta dokonuje odpowiednich operacji na wczytanym obrazie jak np. realokacja. Gdy obraz jest gotowy zostaje przekazany jądru do uruchomienia za pomocą funkcji API. Jądro uruchamia tylko i wyłącznie obraz procesu w formie binarnej gotowej do bezpośredniego przetwarzania przez procesor. Jest to drugi stopień mechanizmu uruchamiania aplikacji.

2.1.1 Kompilacja Asemblera

Aby prawidłowo skompilować aplikacje napisaną w asemblerze należy zastosować się do wcześniej wspomnianej reguły uniwersalnego kodu binarnego przetwarzanego przez kernel. W wersji 0.01 jądra kod wykonywalny procesu jest umieszczany pod adresem 4MB. Stąd, aby skompilować przykładową aplikacje za pomocą NASM'a należy wykonać następujące kroki:

- Umieścić w kodzie następującą informacje dla kompilatora:
```
[BITS 32]
[ORG 0x400000]
```

- Dokonać kompilacji do formatu BIN, np.:
```
C:\>nasm –f bin hello.asm –o hello.bin
```

2.1.2 Kompilacja ANSI C

Kompilacja kodu napisanego w C opiera się na tych samych zasadach co kodu ASM. Poniżej znajduje się przykład kompilacji przy użyciu GCC:

```
C:\>gcc -c hello.c -o hello.o
C:\>ld --oformat binary hello.o -o hello.bin -Ttext=0x400000
```

2.2 Wywołania Systemowe
(API – Application Interface)

Jądro systemu ACD udostępnia aplikacjom zestaw podstawowych procedur. Umożliwiają one komunikację międzyprocesową, zarządzanie swoją pamięcią itd. Wszystkie te procedury (zwane także wywołaniami systemowymi) są dostępne poprzez przerwanie programowe 0x80. Parametry do procedur przekazuje się za pomocą rejestrów ogólnego przeznaczenia. Aby ułatwić pracę, wywołania zostały dodatkowo pogrupowane według kilku kategorii. Numer wywoływanej funkcji zapisuje się w rejestrze AL w następujący sposób: cztery bardziej znaczące bity (hi nibble) to numer grupy, natomiast cztery mniej znaczące bity (lo nibble) to numer funkcji w grupie. Przykładowo AL = 0x31 oznacza, że zostanie uruchomiona pierwsza funkcja z trzeciej grupy. Poniżej znajduje się dokładny opis każdego z dostępnych wywołań.

Grupa 0 – Wywołania ogólne

No.	Wersja	Data	Nazwa
0x00	-	-	-
0x01	1.1	04.05.2007	Wersja systemu
0x02	1.1	04.05.2007	Czas pracy systemu w sekundach
0x03			Czekaj (ms)

Grupa 1 - Procesy

No.	Wersja	Data	Nazwa
0x10			Znajdź proces
0x11	1.1	13.05.2007	Uruchom proces [NFT]
0x12	1.1	13.05.2007	Zakończ proces
			Oddaj resztę jednostki czasu

Grupa 2 – Pamięć

No.	Wersja	Data	Nazwa
0x20	2.1	13.05.2007	Pamięć zajęta przez proces
0x21	1.1	13.05.2007	Powiększenie obszaru danych [NFT]
0x22	1.1	13.05.2007	Pomniejszenie obszaru danych [NFT]
0x23	1.1	13.05.2007	Rozmiar stosu
0x24			Alokuj pod adres
0x25			Alokuj pod adres blok fizyczny
0x26			Przekaż obszar pamięci
0x27			Stan obszaru pamięci

Grupa 3 - Porty

No.	Wersja	Data	Nazwa
0x30	1.1	13.05.2007	Zwolnij wszystkie porty
0x31	1.2	13.05.2007	Alokuj port
0x32	1.2	13.05.2007	Zwolnij port
0x33	1.2	13.05.2007	Alokuj port podwójny
0x34	1.2	13.05.2007	Zwolnij port podwójny
0x35	1.2	13.05.2007	Alokuj port poczwórny
0x36	1.2	13.05.2007	Zwolnij port poczwórny

Grupa 4 – Przerwania

No.	Wersja	Data	Nazwa
0x40			Zamontuj przerwanie
0x41			Zwolnij przerwanie

Grupa 5 – IPC

No.	Wersja	Data	Nazwa
0x50	3.1	07.05.2007	Wyślij wiadomość
0x51	3.1	07.05.2007	Odbierz wiadomość
0x52	3.1	07.05.2007	Odbierz wiadomość z oczekiwaniem
0x53	2.1	07.05.2007	Pobierz ilość wiadomości w skrzynce

2.2.1 Grupa 0 – Wywołania Ogólne

0x01 – Wersja jądra

Funkcja zwraca wersje jądra uruchomionego systemu.

Wejście:

Brak

Wyjście:

EAX – Wersja jądra w postaci:
- Najwyższy bajt – Wysoka wersja
- Niższy bajt – Niska wersja
- Najniższe dwa bajty – Kompilacja

0x02 – Uptime systemu

Zwracaną przez te funkcje wartością jest czas pracy jądra systemu w sekundach (czas od momentu uruchomienia pierwszego procesu do wykonania).

Wejście:

Brak

Wyjście:

EAX – Czas pracy w sekundach.

2.2.2 Grupa 1 – Procesy

0x11 – Uruchom Proces

W celu uruchomienia procesu potomnego, należy załadować jego obraz do pamięci operacyjnej z wyrównaniem adresu początkowego do 4KB (ALIGN 4KB). Następnym krokiem jest utworzenie struktury opisu procesu (zalecane z wyrównaniem adresu początkowego do 4 bajtów) o następującej formie:

Przykład w Assemblerze:

```
proces:
adress            dd ?
size              dd ?
comp              dw ?
verlo             db ?
verhi             db ?
compatible_comp   dw ?
compatible_verlo  db ?
compatible_verhi  db ?
owner             dd ?
name     TIMES 32 db ?
```

Przykład w C:

```
struct process {
unsigned long      adress;
unsigned long      size;
unsigned short int comp;
unsigned char      verlo;
unsigned char      verhi;
unsigned short int compatible_comp;
unsigned char      compatible_verlo;
unsigned char      compatible_verhi;
unsigned long      owner;
char               name[32];
};
```

Znaczenie pól w strukturze:

```
adress            – Adres początkowy obrazu procesu do uruchomienia
size              - Rozmiar obrazu procesu w bajtach
comp              - Numer kompilacji
verlo             - Niska wersja programu
verhi             - Wysoka wersja programu

Pola opisujące najstarszą wersje tego procesu, z którą aktualna wersja jest
zgodna wstecz:

compatible_comp   - Numer kompilacji
compatible_verlo  - Niska wersja programu
compatible_verhi  - Wysoka wersja programu

owner             - Określa UID użytkownika którego mianujemy właścicielem
                    tego procesu jeżeli tworzy go proces roota lub systemu.
                    W przeciwnym razie pole to powinno pozostać wyzerowane.
name              - Ciąg reprezentujący uruchomiony obraz procesu.
```

Wejście:

EBX - Adres struktury opisującej proces

Wyjście:

EAX - Wynik operacji:
0 - Operacja wykonana prawidłowo.
1 - Niewłaściwy adres struktury.
2 - Struktura nie znajduje się w całości w pamięci.
3 - Niewłaściwy adres początkowy procesu.
4 - Niewłaściwe położenie obrazu procesu.
5 - Za mało RAM do uruchomienia procesu.
6 - Obraz procesu za duży.
7 - Obraz procesu nie znajduje się w całości w pamięci.

Kroki wykonania funkcji:

- Sprawdzenie czy adres struktury znajduje się w obszarze danych procesu.
- Jeżeli jest na pograniczu stron następuje sprawdzenie czy obie ramki zawierają się w VPP.
- Przekopiowanie struktury do pamięci tymczasowej jądra.
- Sprawdzenie poprawności struktury:
  - Czy adres początkowy obrazu jest wyrównany do początku ramki.
  - Czy adres początkowy jest w obszarze danych procesu.
  - Czy adres końcowy jest w obszarze danych procesu.
  - Czy istnieje wystarczająca ilość pamięci operacyjnej do uruchomienia procesu.
- Sprawdzenie obecności obrazu procesu w pamięci operacyjnej.
  - Sprawdzenie poprzez badanie obecność tablic stron i samych stron.
- Utworzenie PDB (Process Description Block).
- Podłączenie PDB do kolejki głównej procesów.
- Zainicjowanie PDB danymi startowymi i danymi z struktury.
- Sprawdzenie i ewentualna korekta poprawności pól „wersji zgodnej wstecz".
- Podłączenie PDB do kolejki wykonania.
- Utworzenie VPS.
- Utworzenie stosu procesu.
- Przemapowanie stron z VPS rodzica i odmontowanie ich.

0x12 – Usuń Proces

Usunięcie procesu jest możliwe, jeżeli został on uruchomiony przez tego samego użytkownika, co proces wywołujący funkcje. Odstępstwem od tego jest sytuacja, gdy proces wywołujący funkcje został uruchomiony przez system lub root'a. Ma on wtedy prawo usunąć dowolny proces.

Wejście:

EBX – PID procesu do usunięcia.

Wyjście:

EAX - Wynik operacji:
0 - Operacja wykonana prawidłowo.
1 – Nie ma takiego procesu.
2 – Nie masz praw do usunięcia tego procesu.

2.2.3 Grupa 2 – Pamięć

0x20 – Pamięć zajęta przez proces

Funkcja zwraca ilość pamięci zajmowanej przez proces, oraz ilość pamięci zajmowanej przez struktury systemowe mu przypisane. Oba rozmiary podawane są w bajtach. Uwzględniane są zwolnione obszary w przestrzeni danych pamięci procesu.

Wejście:

brak

Wyjście:

EAX – Ilość pamięci zajmowanej przez proces.
EBX – Ilość pamięci zajmowanej przez struktury systemowe mu przypisane.

0x21 – Powiększenie obszaru danych

Funkcja dokonuje powiększenia obszaru danych procesu o zadany rozmiar (będący wielokrotnością 4KB) poprzez zarezerwowanie brakującej pamięci. Funkcja uwzględnia kolizje z obszarem stosu procesu oraz brak pamięci.

Wejście:

EAX – Rozmiar, o jaki ma być powiększony obszar danych procesu (na 12 niskich bitach ma prawo znajdować się tylko numer wywołania funkcji, inne wartości będą ignorowane).

Wyjście:

EAX – Adres początkowy dołączonego obszaru danych, lub numer błędu:
1 – Brak pamięci operacyjnej do wykonania operacji (fizycznej lub wirtualnej).

0x22 – Pomniejszenie obszaru danych

Funkcja dokonuje pomniejszenia obszaru danych procesu o zadany rozmiar (będący wielokrotnością 4KB) poprzez zwolnienie określonej ilości pamięci z końca obszaru danych procesu. Funkcja uwzględnia brak parametrów lub przekazanie rozmiaru większego od całkowitego rozmiaru obszaru danych.

Wejście:

EAX – Rozmiar, o jaki ma być pomniejszony obszar danych procesu (na 12 niskich bitach ma prawo znajdować się tylko numer wywołania funkcji, inne wartości będą ignorowane).

Wyjście:

EAX – Nowy adres końcowy obszaru danych.

0x23 – Rozmiar Stosu

Funkcja zwraca rozmiar stosu procesu wywołującego tę funkcję.

Wejście:

Brak

Wyjście:

EAX – Rozmiar stosu w bajtach.

2.2.4 Grupa 3 – Porty

0x30 – Zwolnienie wszystkich portów

Funkcja zwalnia wszystkie zarezerwowane przez dany proces porty.

Wejście:

Brak

Wyjście:

Brak.

0x31 – Rezerwacja portu

Zadaniem funkcji jest zarezerwować pojedynczy port dla aplikacji lub w przypadku, gdy port jest zajęty zasygnalizowanie tego kodem wyjścia.

Wejście:

SS:ESP – Numer portu do rezerwacji (dword)

Wyjście:

EAX - Wynik operacji:
- 0 - Operacja wykonana prawidłowo.
- 1 – Port jest już zarezerwowany.
- 2 – Brak wystarczającej ilości pamięci RAM do wykonania operacji.

0x32 – Zwolnienie portu

Funkcja zwalnia pojedynczy port zarezerwowany przez aplikacje lub w przypadku, gdy port jest wolny sygnalizuje to kodem wyjścia.

Wejście:

SS:ESP – Numer portu do zwolnienia (dword)

Wyjście:

EAX - Wynik operacji:
- 0 - Operacja wykonana prawidłowo.
- 1 – Port nie był zarezerwowany lub nie jest zarezerwowany przez te aplikacje.

0x33 – Rezerwacja podwójnego portu

Zadaniem funkcji jest zarezerwować podwójny port dla aplikacji lub w przypadku, gdy port lub jego część jest zajęty zasygnalizowanie tego kodem wyjścia.

Wejście:

SS:ESP – Numer portu do rezerwacji (dword)

Wyjście:

EAX - Wynik operacji:
0 - Operacja wykonana prawidłowo.
1 – Numer portu niewłaściwy.
2 – Port lub jego część jest już zarezerwowany.
3 – Brak wystarczającej ilości pamięci RAM do wykonania operacji.

0x34 – Zwolnienie podwójnego portu

Funkcja zwalnia podwójny port zarezerwowany przez aplikacje lub w przypadku, gdy port lub jego część jest wolny sygnalizuje to kodem wyjścia.

Wejście:

SS:ESP – Numer portu do zwolnienia (dword)

Wyjście:

EAX - Wynik operacji:
0 - Operacja wykonana prawidłowo.
1 – Niewłaściwy numer portu.
2 – Port nie był zarezerwowany, był zarezerwowany częściowo lub nie jest zarezerwowany przez te aplikacje.

0x35 – Rezerwacja poczwórnego portu

Zadaniem funkcji jest zarezerwować poczwórny port dla aplikacji lub w przypadku, gdy port lub jego część jest zajęty zasygnalizowanie tego kodem wyjścia.

Wejście:

SS:ESP – Numer portu do rezerwacji (dword)

Wyjście:

EAX - Wynik operacji:
0 - Operacja wykonana prawidłowo.
1 – Numer portu niewłaściwy.
2 – Port lub jego część jest już zarezerwowany.
3 – Brak wystarczającej ilości pamięci RAM do wykonania operacji.

0x36 – Zwolnienie poczwórnego portu

Funkcja zwalnia poczwórny port zarezerwowany przez aplikacje lub w przypadku, gdy port lub jego część jest wolny sygnalizuje to kodem wyjścia.

Wejście:

SS:ESP – Numer portu do zwolnienia (dword)

Wyjście:

EAX - Wynik operacji:
- 0 - Operacja wykonana prawidłowo.
- 1 – Niewłaściwy numer portu.
- 2 – Port nie był zarezerwowany, był zarezerwowany częściowo lub nie jest zarezerwowany przez te aplikacje.

2.2.5 Grupa 4 – Przerwania

Te funkcje nie są zaimplementowane w chwili obecnej.

2.2.6 Grupa 5 – IPC

0x50 – Wyślij wiadomość

Funkcja wysyła wiadomość z priorytetem do określonego odbiorcy. Jeżeli odbiorca oczekiwał na wiadomość zostaje ona natychmiastowo do niego dostarczona z pominięciem skrzynki odbiorczej, w przeciwnym wypadku, jeżeli skrzynka nie jest przepełniona wiadomość zostaje w niej zapisana.

Wejście:

SS:ESP+12 – Trzeci dword wiadomości
SS:ESP+8 – Drugi dword wiadomości
SS:ESP+4 – Pierwszy dword wiadomości
SS:ESP – Adres odbiorcy, bity 1,0 – priorytet wiadomości

Wyjście:

EAX - Wynik operacji:
0 - Operacja wykonana prawidłowo.
1 – Niewłaściwy adres procesu.
2 – Skrzynka odbiorcy jest pełna.

0x51 – Odbierz wiadomość

Procedura odbiera z skrzynki pocztowej wiadomość od określonego lub dowolnego nadawcy, jeżeli taka istnieje lub zwraca sygnał błędu. Podczas wyboru wiadomości do odczytu, uwzględniane są priorytety wiadomości.

Wejście:

SS:ESP – PID poszukiwanego nadawcy (lub 0xFFFFFFFF czyli dowolny nadawca)

Wyjście:

SS:ESP+12 – Trzeci dword wiadomości
SS:ESP+8 – Drugi dword wiadomości
SS:ESP+4 – Pierwszy dword wiadomości
SS:ESP – PID nadawcy
EAX - Wynik operacji:
0 - Operacja wykonana prawidłowo.
1 – Brak wiadomości.
2 – Brak wiadomości od oczekiwanego nadawcy.

0x52 – Odbierz wiadomość z oczekiwaniem

Procedura odbiera z skrzynki pocztowej wiadomość od określonego lub dowolnego nadawcy, jeżeli taka istnieje lub usypia proces do czasu jej nadejścia. Podczas wyboru wiadomości do odczytu, uwzględniane są priorytety wiadomości.

Wejście:

SS:ESP – PID poszukiwanego nadawcy (lub 0xFFFFFFFF czyli dowolny nadawca)

Wyjście:

SS:ESP+12 – Trzeci dword wiadomości
SS:ESP+8 – Drugi dword wiadomości
SS:ESP+4 – Pierwszy dword wiadomości
SS:ESP – PID nadawcy

Lub

SS:ESP – PID poszukiwanego nadawcy w przypadku uśpienia procesu.

0x53 – Ilość wiadomości w skrzynce

Sprawdza czy w skrzynce są jakieś wiadomości i zwraca ich ilość.

Wejście:

Brak

Wyjście:

EAX – Ilość wiadomości w skrzynce (lub 0 w przypadku ich braku).