2003 유체기계 연구개발 발표회 논문집, pp. 535~540
선문대학교 아산캠퍼스, 2003. 12. 5~6

HVAC를 위한 론기반의 분산형 제어기

최병욱* • 신은철**

LON based Distributed Control System for HVAC

Byoung-Wook Choi*, Eun-Cheol Sin**

Key Words : Lonworks(론웍스), Distributed control(분산제어), HVAC(공조제어), Embedded Linux(임베디드 리눅스)

ABSTRACT

In this paper, a LON based distributed control system for HVAC is described. We developed multi-protocol converter based on SoC, Neuron Chip, embedded Linux. It utilizes the network environment and therefore requires an appropriate operating system for handling protocols and an advanced development environment. The open source licensing, reliability, and broad hardware support are key reasons for use of embedded Linux in embedded industry. The multi-prootocol converter integrates LonWorks devices to a client with Java applet. The system consists of three-tier architecture, such as clients, multi-protocol converter, and LonWorks devices. The experiment result show that multi-protocol converter using embedded Linux is a flexible and effective way to build a Web-based monitoring and control system.

1. 서 론

기존의 거의 모든 제어 장치들은 하나의 호스트에서 모든 제어장치들을 제어하는 중앙 집중형 시스템으로 되어 있다. 이러한 시스템은 폐쇄형 시스템이기 때문에 벤더에 의존적이며, 확장성과 유연성이 떨어진다. 그리고 하나의 시스템에서 모든 작업을 처리하기 때문에 고성의 시스템을 요구한다.

최근 네트워크와 SoC 기술이 발전과 더불어 기존 제어 시스템의 단점을 보완한 실시간 운영체제(RTOS, Real-Time Operating System) 기반의 분산형 제어 장치가 보급되고 있다. 하지만 이러한 시스템도 개발 비용이 비싸고 유연성이 떨어지는 단점이 있다.

본 연구에서는 앞서 설명한 제어 시스템의 단점을 보완하여 론웍스 기반의 분산형 제어 시스템을 개발하고자 한다.

이 시스템을 구현하기 위해 임베디드 리눅스 기반의 멀티-프로토콜 컨버터를 개발한다. 이 장치는 분산형 시스템에서 가장 많이 사용되고 있는 TCP/IP 통신과 론웍스 통신을 지원하며, 사용자 인터페이스를 위해 임베디드 웹서버와 자바 애플릿을 이용하였다. 그림 1은 전체 시스템의 구성을 나타낸다.

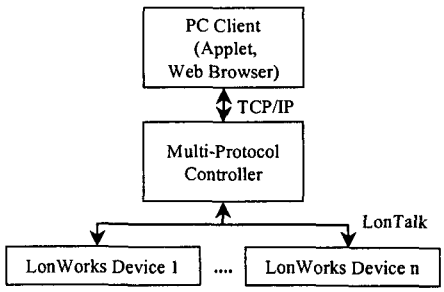

Fig. 1 System block diagram

* 선문대학교 기계및제어공학부
** 선문대학교 기계및제어공학부 대학원
E-mail : bwchoi@sunmoon.ac.kr

2. 멀티-프로토콜 제어기 구성

2.1. 시스템 구성

멀티-프로토콜 컨버터는 2개의 부분으로 구성되어 있다. 첫 번째는 TCP/IP 통신을 위한 부분이며, ARM7TDMI 기반의 삼성 S3C4530A 칩[1]을 이용하였다. 두 번째는 론웍스 통신을 위한 부분이며, Cypress의 CY7C53150 칩을 이용하였다. 이 두 MPU 간의 인터페이스를 위해 DPRAM(Cypress, CY7C136, Dual- Port RAM)을 이용하였다. 주변 장치로는 RS232, LonWorks Port, Ethernet 포트가 있다. 네트워크를 위해 Ethernet 컨트롤러를 사용하였으며, LonWorks를 위해 FTT-10 트랜스시버를 사용하였다. 시스템의 전체적인 구성은 그림 2와 같다.

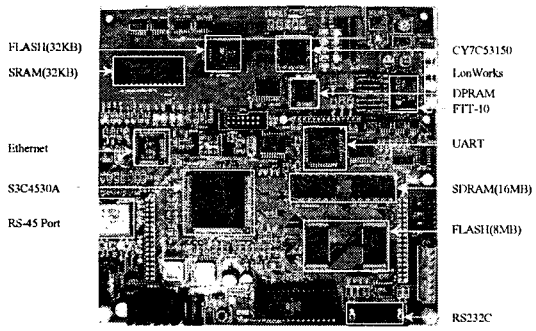

Fig. 2 The component of multi-protocol converter

2.2 리눅스 포팅

리눅스는 기본적으로 32비트 마이크로프로세서인 i386에 처음으로 개발되어 현재는 MMU(Memory Management Unit)이 잇는 여러 마이크로프로세서 아키텍처에 포팅되어 있다. 그러나 본 연구에서 사용하는 S3C4530A는 MMU가 없기 때문에 이러한 기존의 리눅스 커널을 사용할 수 없다. 이와 같이 MMU가 없는 마이크로프로세서들을 위한 임베디드 리눅스인 uClinux를 멀티-프로토콜 제어기의 운영체제로 포팅하였다. uClinux 커널은 일반 리눅스 커널과 달리 가상 메모리 지원을 제거 하였으며, 멀티-타스킹(multi-tasking)을 위한 방법에도 차이가 있다.[3]

가상 메모리를 제거했다는 의미는 하드웨어적으로 이를 지원하지 못하기 때문에, 물리 메모리와 가상 메모리를 같은 주소로 사용한다는 것이다. 따라서 일반 리눅스 커널의 메모리 맵과는 다른게 uClinux 커널은 그림 3과 같이 힙(heap)와 스택(stack)의 위치가 바뀌어 있어 고정된 스택을 사용한다. 멀티-타스킹을 위한 전형적인 리눅스의 fork system call은 copy-on-write라는 기법을 통해 이루어졌지만 uClinux에서는 메모리 사용이 제한적이기 때문에 vfork system call과 함께 thread를 사용한다.

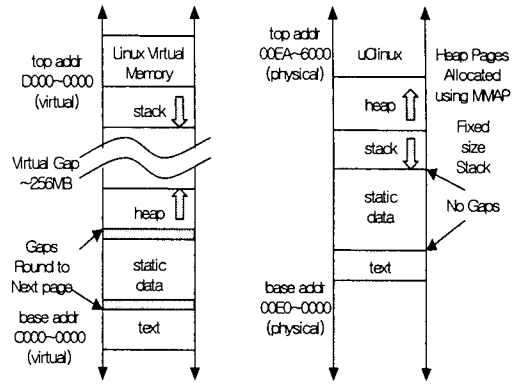

Fig. 3 Memory map of linux and uClinux

uClinux를 포팅하는 과정은 low-level 포팅과 high-level 포팅으로 이루어진다. low-level 포팅은 하드웨어와 관련된 부분으로 다음과 같이 진행된다.[4]

- 엔트리 포인트(entry poing) 정의
- 시스템 레지스터와 PLL 설정
- 시리얼 포트 초기화
- 메모리 맵 초기화
- 스택 초기화
- 프로세서 타입과 머신 파입 설정
- 리눅스 커널(start_kernel)로 점프

low-level 포팅은 어셈블리어로 작성되며, 아키텍처 관련 디렉토리에 위치한다.

high-level 포팅은 리눅스 커널과 관련된 부분으로 low-level에서 호출되어지는 C언어 코드이다. 진행과정은 다음과 같다.

- 리눅스 커널 배널을 출력
- 프로세서와 머신 정보를 설정
- 트랩(exception vector) 초기화
- 인터럽트 초기화
- 스케쥴러에서 요구되는 데이터 초기화
- 소프트웨어 인터럽트 초기화
- 콘솔 초기화
- 인터럽트 인에이블
- 디바이스 드라이버 초기화

2.3 프로세서간의 인터페이스

멀티-프로토콜 컨버터는 2개의 CPU가 존재하기 때문에 이들 사이에 데이터를 인터페이스할 수 있는 방

법이 필요한다. 이를 위한 방법으로 DPRAM(Dual-Port RAM)을 이용하였다. DPRAM은 2개의 데이터 버스와 어드레스 버스를 가지고 있어서 양방향으로 데이터를 읽기/쓰기가 가능하여, 데이터의 변경과 데이터의 동시 쓰기를 방지하기 위해서 인터럽트 신호와 busy 신호를 제공한다. 인터럽트 신호는 저장된 데이터가 변경되었을 때 자동으로 발생하지 않기 때문에 데이터를 변경 후에 인터럽트 발생을 위한 어드레스에 0이 아닌 값을 씀으로써 인터럽트 신호를 만들어 주어야 한다.

3. 로컬 제어기

3.1. 론웍스 디바이스

론웍스(LonWorks)는 BAS에서 표준화되고 있는 통신방법으로 론토크(LonTalk) 프로토콜을 이용한 제어 네트워크를 말하여, 이 프로토콜 스펙을 하드웨어적으로 구현한 프로세서를 뉴런 칩(Neuron Chip)이라 한다.[3] 본 연구에서는 이 칩을 가지고 론 통신 기반의 로컬 제어기를 구현하였다. 론웍스 통신을 위한 트랜스시버(Transceiver)에는 TP(Twist Pair)와 전력선(Power Line), RF(무선통신), 광케이블 등의 종류가 있는데, 그 중 TP 트랜스시버(FTT-10)를 사용하였다. 이 트랜스시버는 76.8Kbps의 통신 속도를 가지며, Free Topology 방식의 배선이 가능하고, 리피터와 같은 외부 장치없이 500m까지 통신이 가능하다. 개발된 론 디바이스의 하드웨어 사양은 표 1과 같다.

표 1 론디바이스 하드웨어 사양

CPU	CY7C53150
Memory	Flash 32KB, SDRAM 32KB
Transceiver	FTT-10
Peripheral	11 GPIOs, Memory-Mapped I/O

이상과 같이 구현된 론 디바이스에 에어컨 제어 회로, 펌프 제어 회로, 온도 센서 회로, 조도 센서 회로, 7-세스먼트 회로, 발광다이어드 회로를 추가하여 그림 4과 같이 구현하였다.

각각의 제어기는 뉴런 C(Neuron C)를 이용하여 프로그램하였다. 뉴런 C는 뉴런칩을 개발하기 위한 언어로써 ANSI C를 기반으로 만들어졌으며, 네트워크와 I/O 포트를 위한 데이터 타입이 추가되었다. 또한 이벤트-핸들링을 지원하기 때문에 이벤트 방식의 프로그램이 가능하다.

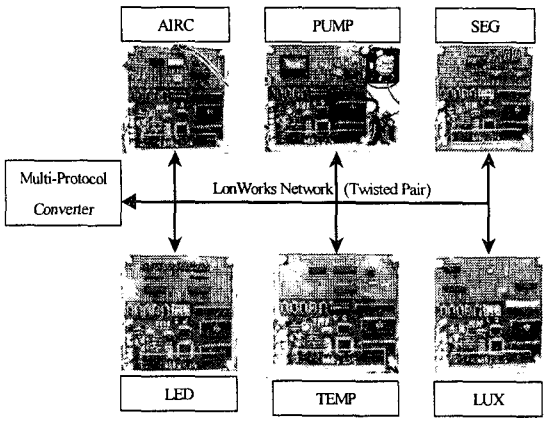

Fig. 4 LonWorks network to control devices

각각의 론 디바이스 그림 5와 같이 *reset()*, *handleEvent()*, *handleProcess()*(handleEvent와 handle Process() 함수는 일반화한 이름이다.) 구성된다. *reset()*은 디바이스에 전원이 인가되면 가장 먼저 실행되는 함수이다. 이 함수에서는 디바이스를 초기화하는 로직이 들어가게 된다. *handleEvent()* 함수는 이벤트가 발생했을 때 실행되는 함수로 실행되는 함수로, 네트워크 변수의 값이 변경되거나 타이머 오버플로우가 발생하면 실행되도록 설정하였다. *handleProcess()* 함수는 *handleEvent()* 함수에서 이벤트의 종류 및 값에 따라 실행되는 함수이다. 이 함수에서는 I/O 포트와 Memory-Mapped I/O를 이용하여 인터페이스 회로를 제어하는 신호를 만든다.

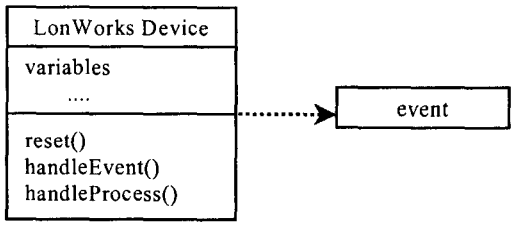

Fig. 5 S/W architecture of LonWorks devices

3.2 네트워크 바인딩

앞서 각각의 론 디바이스를 개발하였지만 아직 서로 론웍스 네트워크로 연결된 것은 아니다. 이것들을 서로 연결하기 위해서는 바인딩(binding)이라는 작업이 필요하다. 바인딩은 론웍스 통신으로 연결된 론 디바이스들을 연결하는 작업이다. 바인딩을 하기 위해서는 우선 네트워크 변수나 네트워크 태그를 선언해주어야 한다. 본 연구에서는 네트워크 변수만을 사용하였기

때문에 이부분에 대하여만 논하도록 하겠다. 네트워크 변수는 반드시 입력과 출력의 데이터 타입이 같아야 하며, 입출력이 1:1 또는 1:n으로 연결되어야 한다. 그림 6은 로컬 제어기들의 바인딩 구조를 나타낸다.

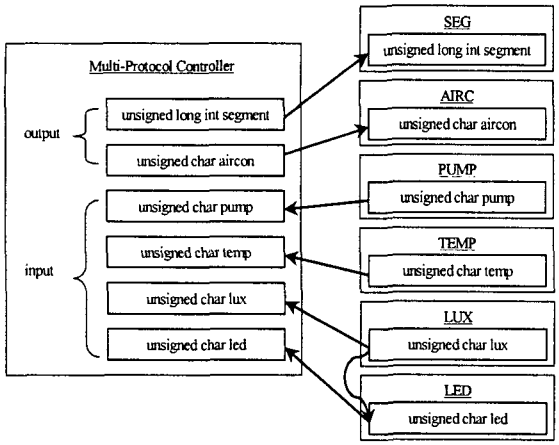

그림 6. Structure of binding of network variables

각각의 네트워크 변수에 대하여 알아보겠다. 멀티-프로토콜 컨버터는 각각의 로컬 제어기와 1:1로 연결되어 있고, 로컬 제어기 중 *luxmeter*와 *leds*만 서로 연결되어 있다. *segment*는 7-세그먼트에 출력될 값과 어떤 7-세그먼트를 제어할지에 대한 정보를 가지고 있고, *aircon*는 에어컨 제어에 필요한 명령을 가진다. *pump*는 펌프의 On/Off 상태값을, *temperature*와 *luxmeter*는 온도센서와 조도센서의 A/D 컨버팅된 값을 저장한다. *leds*는 *luxmeter*의 값을 받는다.

4. 프로토콜 변환 알고리즘

멀티-프로토콜 컨버터는 TCP/IP를 통해 론웍스 기반의 로컬제어기를 제어하고 모니터링한다. 이 장에서는 멀티-프로토콜 컨버터의 서버 프로그램의 구조와 어떻게 프로토콜을 변환하는지 알아보겠다.

서버 프로그램의 구조는 그림 7에서와 같이 *LON*과 *Server*로 구성된다. *LON* 부분은 론 디바이스를 제어하는 모듈이며, *Server*는 TCP기반의 소켓 서버 모듈으로써 프로토콜을 변환하는 부분이 포함되어 있다.

서버 프로그램의 동작 순서는 다음과 같다.

1) 소켓 서버를 생성하고 클라이언트의 접속을 기다린다.

2) 클라이언트에서 접속하면 새로운 쓰레드를 생성하여 클라이언트와 통신을 한다.

3) 클라이언트로부터 받은 데이터를 분석하여 해당되는 로컬제어기를 제어한다.

4) 클라이언트로부터 종료 메시지를 받거나 접속이 끊기면 생성된 쓰레드를 소멸시키고 다시 클라이언트의 접속을 기다린다.

이번에는 데이터 패킷의 구조와 어떤 식으로 로컬 제어기에 명령을 내리는지 설명하겠다. 서버 프로그램은 그림 7과 같은 패킷 구조로 되어 있다.

recv packet

'$'	addr	cmd	dataL	dataH	'#'

send packet

'$'	pump	temp	lux	led	'#'

Fig. 7 The packet of TCP/IP

'$'와 '#'은 데이터의 시작과 종료를 나타낸다. *addr* 필드는 어떤 로컬 제어기를 명령을 내릴것인지 상태값을 가져가는지를 나타내며, *cmd* 필드는 명령을, *dataL*과 *dataH*는 명령에 대한 부가적인 정보를 포함한다. TCP/IP는 자체적으로 에러 체크 로직을 가지고 있으므로 에러 체크 부분을 제외시켰다. *pump*와 *temp*, *lux*, *led* 필드는 각 장치의 상태값을 가진다.

*addr*에 로컬 제어기를 제어하는 명령이 들어오면 *cmd*, *dataL*, *dataH*를 DPRAM에 그림 8과 같이 저장한 후, 인터럽트 신호을 발생시키기 위해 0x7f2 번지에 0이 아닌 값을 저장한다. 그러면 뉴런 칩은 인터럽트 신호를 받아서 이벤트를 발생시킨다. *eventHandle()* 함수에서는 DPRAM에 저장된 값을 읽어서 해당되는 로컬 디바이스와 연결된 네트워크 변수에 값을 대입함으로써 명령을 내린다. 상태값을 가져오는 방법은 명령을 내리는 방식과는 다른 알고리즘을 사용한다. 각각의 로컬 디바이스는 자신의 상태를 1초에 한번씩 멀티-프로토콜 컨버터로 보낸다. 데이터를 받은 멀티-프로토콜 컨버터는 DPRAM에 이 값들을 저장하고 있다가 상태를 요구하는 명령을 받으면 DPRAM에서 상태값을 읽어서 보내준다.

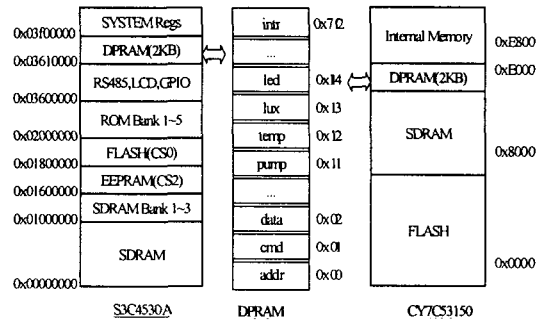

그림 8. The data transfer mechanism between CPUs

5. 론기반의 분산형 시스템 구현

5.1. 시스템 구성

그림 9는 구성된 시스템의 구조를 나타낸다.

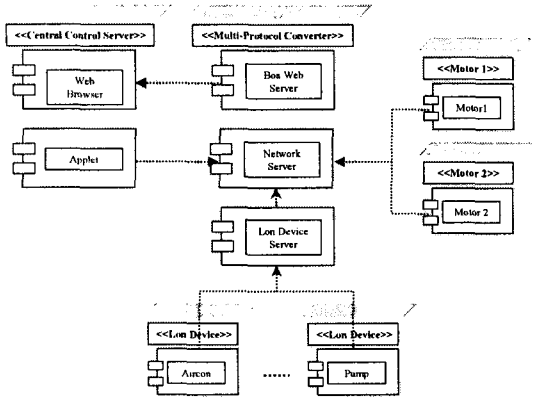

Fig. 9 The system architecture

일반 PC에서 멀티-프로토콜 컨버터로 접속하기 위해 임베디드 웹서버를 임베디드 리눅스에 포팅하였으며, 사용자 인터페이스를 위한 GUI로는 자바 애플릿을 사용하였다. 자바 애플릿는 서버 스크립트 언어와는 달리 클라이언트에서 클라이언트에서 실행되고 임베디드 웹서버에서도 동작될 뿐만 아니라 자바 버츄얼 머신(VM, Virtual Machine)만 있으면 OS와 상관없이 실행되는 장점이 있다.

본 연구에서 사용된 애플릿 프로그램의 구조는 그림 10과 같다.

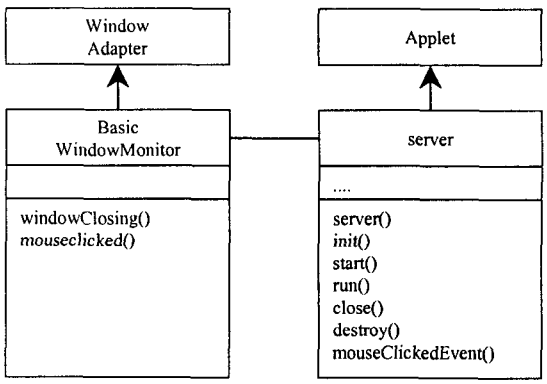

Fig. 10 The structure of java applet program

이 프로그램은 클라이언트로 다운로드되면 init() 함수가 가장 먼저 실행된다. 이 함수는 사용되는 변수를 초기화하고 GUI 화면에 출력한다. 그 다음 start() 함수가 실행되는데, 이 함수는 멀티-프로토콜 컨버터의 서버 프로그램과 접속을 하고 TimerTask를 실행한다. TimerTask는 1초 간격으로 제어기의 상태를 모니터링하기 위해서 사용된다. run() 함수는 TimerTask에서 실제로 실행되는 함수이다. 로컬 제어기를 제어하기 위해 제어 버튼을 클릭하면 WindowMonitor로부터 mouseClicked 이벤트가 발생하여 각 버튼에 따른 mouseClickedHandle 함수가 실행된다. 이 함수는 앞에서 설명한 멀티-프로토콜 컨버터의 서버 프로그램의 데이터 패킷 형태에 맞게 데이터를 만들어서 보낸다. destroy() 함수는 프로그램이 종료될 때 실행되는 함수이며 서버에 프로그램이 종료됨을 알리고 클라이어트 소켓을 받는다. 그림 11은 이상과 같이 구현된 자바 애플릿의 GUI이다.

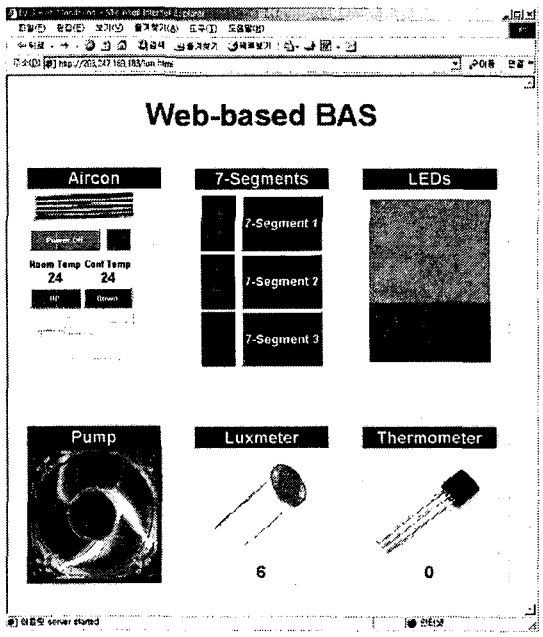

Fig. 11 GUI implemented in java applet

6. 결 론

본 연구에서 임베디드 리눅스와 임베디드 웹서버 기반의 멀티-프로토콜 컨버터를 개발하여 HVAC를 위한 론웍스 기반의 분산형 시스템을 구현하였다. 멀티-프로토콜 컨버터의 운영체제로 상용 실시간운영체제가 아닌 임베디드 리눅스를 채택하였으며 TCP/IP와 론웍스를 지원하기 위해 ARM7TDMI 기반의 SoC와 뉴런칩을 사용하였으며, 이들 사이의 인터페이스를 위해 DPRAM을 이용하였다.

멀티-프로토콜 컨버터는 사용자 인터페이스를 자바 애플릿으로 구현함으로써 운영체제와 무관하게 시스템을 모니터링할 수 있으며, 다른 제어 장치와 TCP/IP

표준 프로토콜을 이용하여 데이터를 송수신하는 것이 가능하기 때문에 시스템을 유연하게 구성할 수 있다.

참고문헌

(1) http://samsung.com/Products/Semiconductor/SystemLSI/Networks/PersonalNTASSP/CommunicationProcessor/S3C4530A/S3C4530A.htm, "S3C4530A User's Guide"

(2) http://www.uclinux.ort/description, Embedded Linux Microcontroller Project

(3) http://www.echelon.com/product/lonworks/default.html, "Introduction to the Lonworks system"

(4) 최병욱, 고경철, 문전일, 임계영, 2003, 임베디드 리눅스 실습 및 활용, 홍능과학출판사, 서울, pp. 208-210.

(5) 이현우, 천영환, 2001, Java Programming Bible for JDK1.3, 영진.com, 서울, pp. 869-900.