

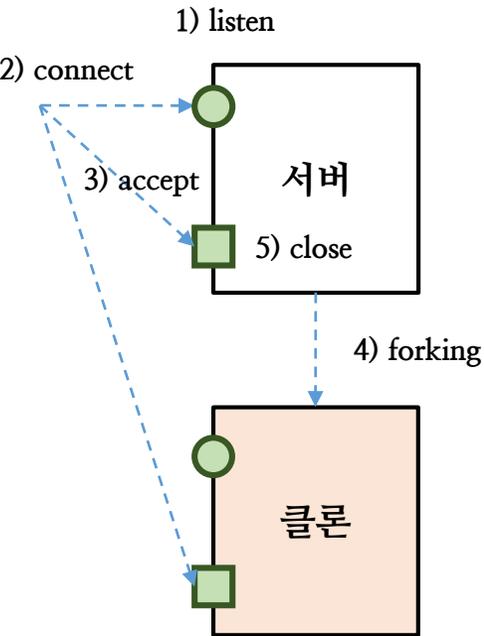
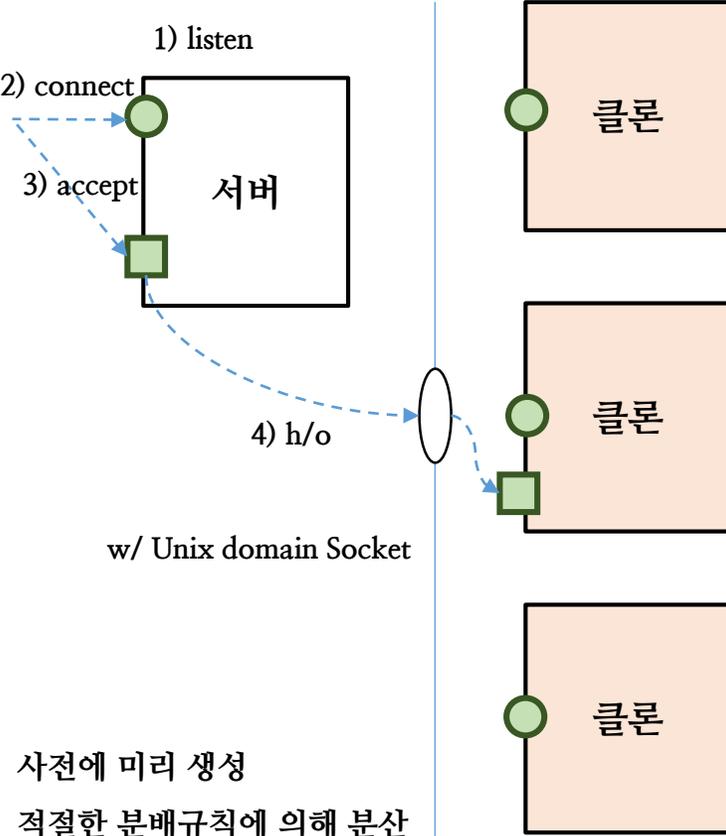
영상플랫폼 구조개선 적용기술

2015/06/05

(주) 엠앤엘솔루션

부하분산

지정된 특정포트로 “스트리밍”하는 서버의 부하분산 방법으로 전처리 클로닝 & 소켓이관

기존	개선방향	
 <p>서비스 별 forking하는 구조로서 자원소모가 많음 (전체복사; 메모리, CPU) 시간지연요소 발생</p>	 <p>사전에 미리 생성 적절한 분배규칙에 의해 분산 연결된 소켓정보를 이관</p>	<pre> int handover (int ufd, void *ptr, size_t nbytes, int fd, char *path) { struct sockaddr_un sun; int val; struct msghdr msg; struct iovec vec; union { struct cmsghdr cm; char control[CMMSG_SPACE(sizeof(int))]; } control_un; struct cmsghdr *cmptr; sun.sun_family = AF_UNIX; strcpy(sun.sun_path, path); msg.msg_name = &sun; msg.msg_namelen = sizeof(sun); vec.iov_base = ptr; vec.iov_len = nbytes; msg.msg_iov = &vec; msg.msg_iovlen = 1; msg.msg_control = control_un.control; msg.msg_controllen = sizeof(control_un.control); msg.msg_flags = 0; cmptr = CMSG_FIRSTHDR(&msg); cmptr->cmsg_level = SOL_SOCKET; cmptr->cmsg_type = SCM_RIGHTS; cmptr->cmsg_len = CMSG_LEN(sizeof(int)); *((int *) CMSG_DATA(cmptr)) = fd; return(sendmsg(ufd, &msg, 0)); } </pre>

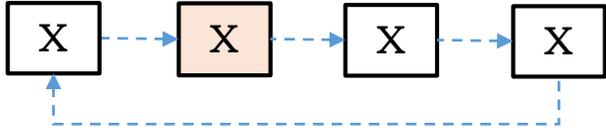
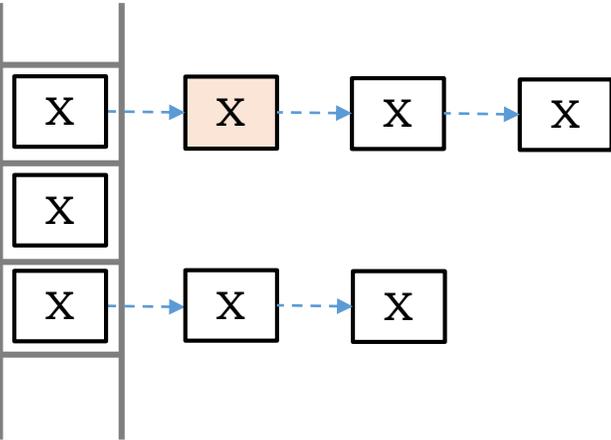
프로세스 경량화를 통한 고 용량 시스템

가입자 서비스 단위 별 프로세스 생성방식에서 그룹별 프로세스 생성/관리, 동일자원대비 고집적 구현

기존	개선방향	해시함수																																			
<div data-bbox="163 464 789 835" data-label="Diagram"> <pre> graph TD S[서버] -.-> C1[클론] S -.-> C2[클론] S -.-> C3[클론] </pre> </div> <p data-bbox="132 902 810 1110"> 가입자 서비스 별로 프로세스를 생성하므로 서버자원을 많이 소모하고 서버에 관리/분배역할 집중, 서버가 재기동할 경우 이전의 프로세스를 관리할 수 없음. </p>	<div data-bbox="912 464 1592 863" data-label="Diagram"> <pre> graph TD S[서버] -.-> G1[그룹] S -.-> G2[그룹] S -.-> G3[그룹] </pre> </div> <p data-bbox="912 902 1592 1225"> 가입자와 관련된 키(KEY)를 기반으로 분배, 서버는 빠르게 부하분산에 집중 최소한의 관리정보를 취하고 그룹에서 관리 서버 재 기동 이후 관리 가능 서버는 그룹의 셸 (SHELL)로 동작하며, 그룹 프로 세스의 이상 상황 감지 시 재기동 작업을 수행 </p>	<div data-bbox="1676 406 2407 649" data-label="Diagram"> </div> <div data-bbox="1694 664 2382 1178" data-label="Table"> <table border="1"> <thead> <tr> <th></th> <th>ISA</th> <th>OSA</th> <th>영속성</th> <th>VOC</th> </tr> </thead> <tbody> <tr> <td>ter_id</td> <td>○</td> <td>○</td> <td>○</td> <td></td> </tr> <tr> <td>ter_token</td> <td>○</td> <td>○</td> <td></td> <td></td> </tr> <tr> <td>user_id</td> <td></td> <td>○</td> <td>○</td> <td>○</td> </tr> <tr> <td>user_token</td> <td></td> <td>○</td> <td></td> <td></td> </tr> <tr> <td>mac</td> <td>○</td> <td></td> <td>○</td> <td>○</td> </tr> <tr> <td>RTSP URL</td> <td></td> <td>○</td> <td></td> <td></td> </tr> </tbody> </table> </div> <p data-bbox="1694 1206 2407 1306"> 내부 세션 관리를 위한 호환키로서 ter_id 고객응대관련 키로 {user-id mac} 사용 </p>		ISA	OSA	영속성	VOC	ter_id	○	○	○		ter_token	○	○			user_id		○	○	○	user_token		○			mac	○		○	○	RTSP URL		○		
	ISA	OSA	영속성	VOC																																	
ter_id	○	○	○																																		
ter_token	○	○																																			
user_id		○	○	○																																	
user_token		○																																			
mac	○		○	○																																	
RTSP URL		○																																			

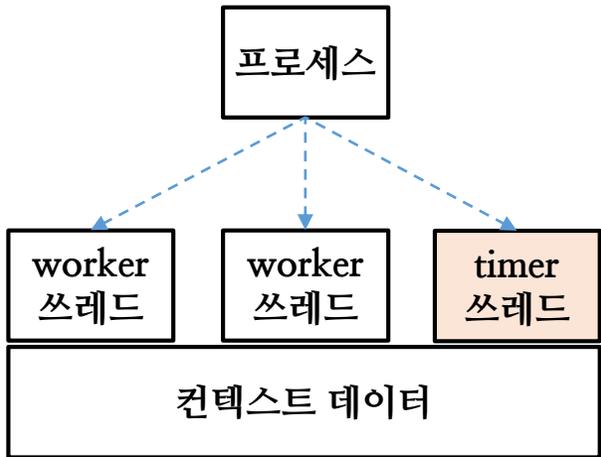
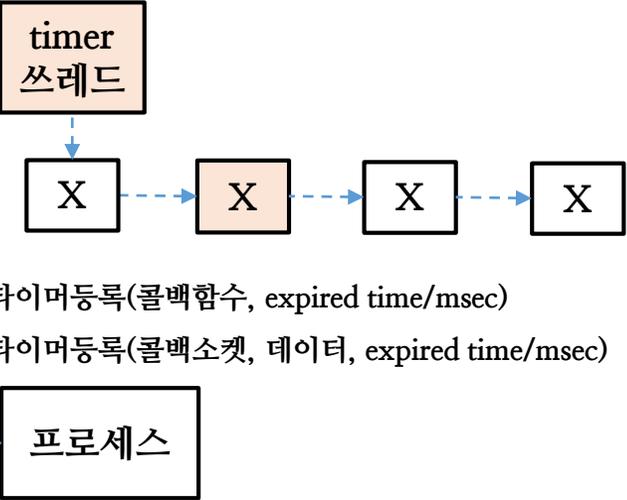
실시간 자료색인 및 검색속도 개선

메모리, 오픈 해시 기반 빠른 검색

기존	개선방향	
 <p>환형 큐 방식의 linked-list 구조체 사용 Pending 자료(대부분 대기열 소켓)가 많아질 경우 급격한 성능 저하 발생</p>	 <p>해시 array에 linked-list를 연결한 방식 해시함수를 이용, 적절히 분배하면 O(1) 검색속도 예) array 크기가 256개에 linked-list depth가 4이면 1024 가입자의 빠른 검색이 가능, 자원손실 최소</p> <p>적용된 linked-list 기법은 Linux 커널에서 사용 검증된 "list.h" 사용</p>	<pre>#define sockHashFunc(d) ((d) % MAX_HASH) #define keyHashFunc(d,l) (atoi(&(d)[0]+1-4)%MAX_HASH) typedef struct { pthread_mutex_t lock; uint8_t priv[0]; uint8_t mac [MAC_LEN+4]; uint8_t ter_id [TER_ID_LEN+4]; uint8_t ter_token [TER_TOKEN_LEN+4]; uint32_t user_id; uint8_t user_token [USER_TOKEN_LEN+4]; char url[SHORT_URL_LEN+4]; // URL char sdp[BUFSIZ]; // SDP int retry; int up_bytes; int up_packets; int dn_bytes; int dn_packets; struct list_head til; // ter_id struct list_head ttl; // ter_token struct list_head uil; // user_id struct list_head utl; // user_token struct list_head csl; // control socket struct list_head ssl; // stream socket struct list_head rsl; // relay socket struct list_head tsl; // trigger socket sockh_t *csock; // control sockh_t *ssock; // stream sockh_t *rsock; // relay (ISP <-> OSP) sockh_t *tsock; // trigger (EB) } keyh_t; // 자료예시</pre>

효율적인 타이머기술 적용

주기적 작업, 재전송 등의 작업에 효율적인 전역타이머 쓰레드 적용

기존	개선방향	
<p>기존 서버에서의 Keep-alive / call-check 구현 예시)</p>  <p>프로세스 별 타이머 쓰레드 존재, worker thread는 모든 메시지의 시각을 컨텍스트에 남기고, 타이머 쓰레드가 무한루프를 돌면서 현재의 시각과 비교하는 방식으로 구현</p>	 <p>타이머등록(콜백함수, expired time/msec) 타이머등록(콜백소켓, 데이터, expired time/msec)</p> <p>독립적인 타이머 thread 존재, 등록 함수에 따라 timer가 직접 콜백함수를 실행 또는 콜백 소켓으로 데이터를 만료시간에 전달해 줌 최적성능을 위하여 등록 시, 만료시간을 기준으로 Sorting하고 timer thread가 일정주기로 wake하여 현재시간 이전 작업을 순차적으로 처리</p>	<pre>#define MAXCB 10 void cbfn(char* d) { struct timeb t; ftime(&t); printf("%d.%03d : (-) %s \Wn",t.time,t.millitm,d); } int main() { int tid[MAXCB],i; char p[MAXCB][20]; struct timeb t; for(i=0;i<MAXCB;i++) { ftime(&t); sprintf(p[i], "Hello-%02d",i); tid[i] = set_cb_timeout (cbfn, p[i], 1000*(i+1)); printf("%d.%03d : (+) %s (callback after %7.3f sec)\Wn", t.time,t.millitm,p[i],(float)1000*(i+1)/1000); } while(1) sleep(1); } </pre>

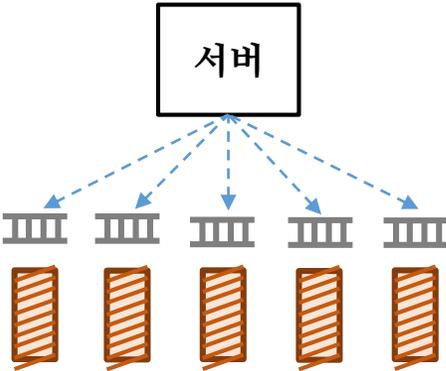
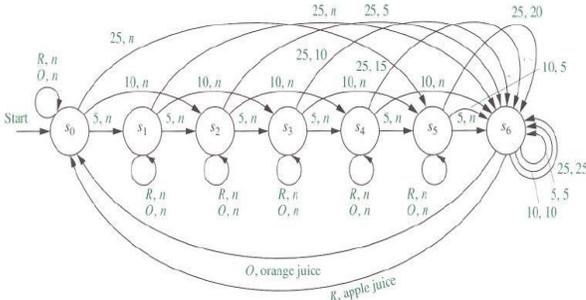
인터페이스 추상화 모델 및 개발관리 방안 제시

스트리밍 서버 구성요소인 *inbound/outbound streamer* 인터페이스 추상화 제시

기존	개선방향	
<div data-bbox="270 445 670 731" data-label="Diagram"> </div> <p data-bbox="142 805 817 901">재 사용가능 공통모듈 없음 개발 방법론 부재에 따른 프로그램 가독 성 떨어짐</p>	<div data-bbox="947 445 1579 668" data-label="Diagram"> </div> <p data-bbox="907 708 1595 858">실제 각 프로세스는 대부분 논리 코드 만 존재, 작업의 대부분을 공용화하여 라이브러리 화 전체 스트리머의 인터페이스 추상화모델 개발/구현</p> <div data-bbox="996 893 1523 1358" data-label="Diagram"> </div>	<p data-bbox="1676 337 2066 365">각 프로세스 파일 구조 (모두 동일)</p> <ul data-bbox="1676 396 2351 596" style="list-style-type: none"> <i>main.c</i> 메인함수 <i>init.c</i> 초기화및 종료관련 (자료형/DB/로그/환경설정/시그널관리) <i>cock.c</i> 논리코드 (소켓관련 처리) <i>command.c</i> 커맨드, 명령어처리관련 <i>Fsm.h</i> 유한상태기계 (Finite State Machine) 정의 및 콜백함수지정 <i>config.c</i> 환경설정 수행부 <i>config.h</i> 환경설정 정의 <p data-bbox="1676 654 1862 682">라이브러리 구성</p> <ul data-bbox="1676 715 2288 1082" style="list-style-type: none"> <i>db.c</i> 데이터베이스 ODBC wrapper/API 함수 <i>hash.c/h</i> 스트리밍관련 해시함수 <i>list.h</i> 리눅스커널 함수를 사용자스페이스로 포팅 <i>cctv.c/h</i> CCTV 관련 API/라이브러리 <i>rtsp.c/h</i> RTSP관련 클라이언트/서버 API 함수 <i>timefmt.c/h</i> 각종 시각포맷변환 함수 <i>sock.c/h</i> 소켓관련 API 함수 <i>seed.c/h</i> 암호화관련 함수 <i>premmc.c / mmc.C</i> 커맨드라인 기반 명령어처리 파서/함수 <i>rc.c/h</i> 서버자원관리 함수 <i>proc.c/h</i> 프로세스관리함수 <i>log.c/h</i> 로그/TLO 관련함수 <i>util.h</i> 각종 매크로 유틸리티

fsm (finite state machine) 을 적용한 상태천이관리

복잡한 논리단계를 코드에 쉽게 적용가능

기존	개선방향	유한상태관리기 자료형 : 특정상태에서 이벤트를 수신하면 해당 콜백함수가 수행되는 구조
<p>클래스기반의 복잡한 논리구조로 설계 구현 각 메시지 별로 클래스 정의 기능단위 별로 워크 thread가 작성되어 있으므로, IPC (각 객체/클래스/쓰레드 별로 고유의 메시지 큐 및 내부버퍼를 사용)의 단계가 복잡하고 지연요소가 발생</p> 	 <p>협상 메시지 송/수신 기반 시스템에서 유한상태기계를 사전에 정의하고, 상태와 이벤트에 의해서 수행할 작업(action)과 상태천이를 수행하도록 설계/구현함</p>	<pre>typedef struct { int status; int event; int (*cbFunc)(void*, void*); int next; } fsm_t; ISP/OSP에서 Relay negotiation하는 유한상태기계 자료정의 static fsm_t rfsm[] = { {OPEN, UP, cbSendTypeReq, SENT(REQ(SEND_TYPE))}, {SENT(REQ(SEND_TYPE)), RCVD(RES(SEND_TYPE)), NULL, RCVD(RES(SEND_TYPE))}, {RCVD(RES(SEND_TYPE)), RCVD(REQ(SDP)), cbUserTokenReq, SENT(REQ(USER_TOKEN))}, {SENT(REQ(USER_TOKEN)), RCVD(RES(USER_TOKEN)), NULL, RCVD(RES(USER_TOKEN))}, {RCVD(RES(USER_TOKEN)), RCVD(REQ(STRM_START)), cbStartStrmResp, ESTABLISH}, {ESTABLISH, RCVD(REQ(STRM_START)), cbStartStrmResp, ESTABLISH}, {ESTABLISH, RCVD(REQ(STRM_STOP)), cbStopStrmResp, ESTABLISH}, {0, 0, NULL, 0} };</pre>